VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

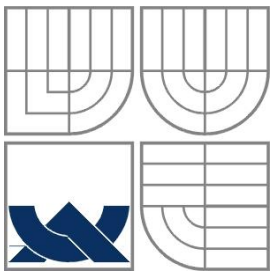
ZVYŠOVÁNÍ RYCHLOSTI MODERNÍCH WEBOVÝCH APLIKACÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

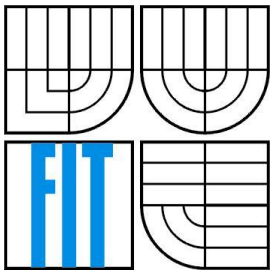
AUTOR PRÁCE
AUTHOR

Bc. RADEK ČEPL

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZVYŠOVÁNÍ RYCHLOSTI MODERNÍCH WEBOVÝCH APLIKACÍ

ACCELERATION OF MODERN WEB APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. RADEK ČEPL

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Dr. Ing. PAVEL ZEMČÍK

BRNO 2011

Zvyšování rychlosti moderních webových aplikací

Acceleration of Modern Web Applications

Vedoucí:

Zemčík Pavel, doc. Dr. Ing., UPGM FIT VUT

Oponent:

Mlích Jozef, Ing., UPGM FIT VUT

Student:

Čepl Radek, Bc.

Zadání:

1. Prostudujte literaturu na téma "rychlost webových aplikací" a to zejména se zaměřením na návrh aplikací a jejich vyhodnocování.
2. Analyzujte možnosti zvyšování a měření rychlosti vybraných druhů webových aplikací.
3. Vyberte vhodnou pokročilou techniku pro budování moderních webových aplikací, přitom se zaměřte zejména na svobodné SW platformy.
4. Využijte vybranou techniku urychlení aplikací. Změřte zvýšení rychlosti.
5. Diskutujte dosažené výsledky a možnosti pokračování práce.

Část požadovaná pro obhajobu SP:

Body 1 až 3 zadání.

Kategorie:

Uživatelská rozhraní

Literatura:

- dle pokynů vedoucího

Komentář:

Vypsáno po dohodě s M. Bezsedesem, CAMEA.

Abstrakt

Diplomová práce se zabývá funkcí a strukturou webových aplikací, popisuje jednotlivé technologie používané v těchto aplikacích. Rovněž vysvětluje, jakým způsobem se vytváří a to za účelem vysoké efektivity a snadného vývoje. Hlavní část práce představuje technologie ke zrychlení těchto aplikací, vysvětluje jejich nastavení a vlastnosti. V závěru jsou technologie důkladně otestovány, zhodnocen přínos při použití a doporučeno uplatnění pro budoucí vývoj.

Abstract

The thesis deals with function and structure of web applications, describes the individual technologies used in these applications. It also explains how to create for the purpose of high efficiency and easy development. The main part presents technologies to speed up the applications, explain their settings and properties. Finally, the technologies are thoroughly tested, evaluated benefits of use and recommended the application for future development.

Klíčová slova

Webová aplikace, Databáze, Zend Framework, AJAX, Kešování, Komprese, Jmeter

Keywords

Web application, Database, Zend Framework, AJAX, Caching, Compression, Jmeter

Citace

Radek Čepl: Zvyšování rychlosti moderních webových aplikací, Brno, FIT VUT v Brně, 2011

Zvyšování rychlosti moderních webových aplikací

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením doc. Dr. Ing. Pavla Zemčíka.
Další informace mi poskytl Ing. Marian Beszédeš, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Bc. Radek Čepl
25. 5. 2011

Poděkování

Za vedení při práci děkuji doc. Dr. Ing. Pavlovi Zemčíkovi a za odborné rady a pomoc Ing. Marianovi Beszédešovi, Ph.D.

Obsah

1 Úvod

Dnešní svět by nemohl fungovat bez stále se rozvíjejících technologií, které velkou měrou ovlivňují naše životy. Ve světě výpočetních technologií jsou to bezesporu počítače ve všech jejich podobách. Ty už se nepoužívají pouze k jednoduchým úkonům, ale často na nich závisí i celé rozsáhlé systémy. Zde všude už existují sítě počítačů, aby spolu jednotlivé části systému mohly komunikovat a celý proces správy byl rychlejší a efektivnější.

Největší a nejznámější síť dnešního světa je bezesporu internet. Je k němu připojeno obrovské procento lidí, spojuje téměř každého s každým a stal se tak obrovským médiem pro uchování a přenos informací, že na něm lze nalézt nebo pomocí něj zpracovat už téměř cokoliv.

Programy běžící na internetu a obsluhující masy uživatelů se stávají čím dál oblíbenější, chytřejší a propracovanější. Komunikují spolu přes síťový protokol HTTP a nazývají se webové aplikace. Tyto aplikace jsou používané ve všech oborech, pro nejrůznější úkony a pomalu nahrazují samostatné aplikace pro konkrétní počítač. Používají se především kvůli rychlé komunikaci, dostupností odkudkoliv a snadnému sdílení prostředků.

Ovšem s popularitou webových aplikací přichází i problémy v podobě velkého počtu uživatelů a z toho plynoucí potřeby přenést čím dál větší množství dat. Aby mohly být všechny požadavky uspokojeny, uživatelé mohli plynule pracovat, organizace nepřicházely o zisky a nemusel se nikdo obávat o funkčnost podstatných řídicích systémů, musí se tyto aplikace neustále zrychlovat a optimalizovat.

Tato práce popisuje funkci a strukturu moderních webových aplikací, na jakých technologiích jsou postaveny a jakým způsobem se vytváří. Představeny jsou jak technologie na straně klienta, tak na straně serveru.

V první části je věnována pozornost úvodu do problematiky a představení základních celků webových aplikací, které spolu musí spolupracovat a jsou jejich nedílnou součástí.

V další části práce se nachází analýza míst, kde lze aplikaci zrychlovat a představeny možnosti k realizaci tohoto cíle. Vybrány byly čtyři aktivní technologie, pomocí kterých lze dosáhnout požadovaného zrychlení a které jsou hojně využívány v moderních webových aplikacích.

Závěr práce obsahuje popis implementace testovací aplikace, testy vybraných technologií ke zrychlení, popis měřící aplikace a zpracování výsledků měření.

2 Webové aplikace

Tato kapitola uvádí do problematiky webových aplikací a věnuje se shrnutí současného stavu na poli tohoto typu aplikací. Jako základ pro vysvětlení jsou použity pouze ty poznatky, které jsou třeba pro další části této práce.

Kapitola 2.1 popisuje, čím webové aplikace byly a jak se změnila jejich funkce. V kapitole 2.2 je představen webový server jakožto jeden ze základních prvků, vysvětleno jakým způsobem obsluhuje požadavky a jakou má strukturu. Strukturu vlastní webové aplikace a ukázku, jak vypadá, obsahuje kapitola 2.3. Kapitola 2.4 vysvětluje funkci databáze ve webových aplikacích a popisuje její vlastnosti.

Webová aplikace je aplikace typu klient-server běžící na webovém serveru a ke které uživatelé přistupují pomocí počítačové sítě. Komunikace probíhá odesíláním odpovědí serveru na příchozí požadavky. Pro práci s takovýmto typem aplikací je zapotřebí pouze tenký klient v podobě webového prohlížeče.

Jelikož se samotná aplikace skládá z mnoha vzájemně komunikujících elementů, opět platí, že jak je rychlý nejslabší článek, tak je rychlý celek. Aby byly požadavky obslouženy a vráceny klientovi co nejrychleji, musí být aplikováno mnoho pomocných technik, které tyto slabá místa eliminují a napomohou tak k rychlejšímu chodu celé aplikace.

Jednotlivé technologie, na kterých jsou webové aplikace postaveny a pomocí nichž budou implementovány pomocné techniky, představuje kapitola 3.

2.1 Historie

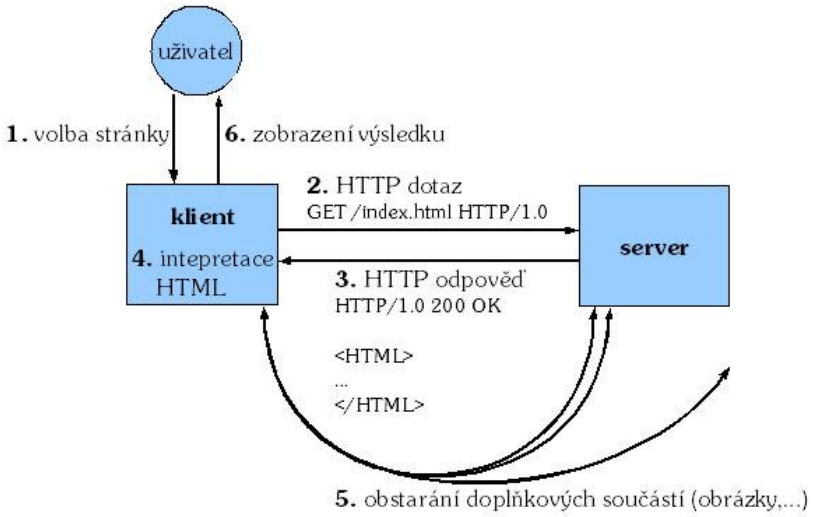
Dalo by se říct, že současné webové aplikace navázaly na dřívější klient-server aplikace, které musely mít vlastního klienta. Tohoto klienta bylo zapotřebí nainstalovat na počítač každého uživatele a sloužil jako uživatelské rozhraní. Problém těchto aplikací bylo, že při aktualizaci na straně serveru se rovněž musely aktualizovat klientské programy na každém uživatelském počítači. To bylo velmi neefektivní a nákladné.

Koncept webové aplikace tyto problémy definitivně odstranil. Jako výstup aplikace je předložen dokument, který prohlížeč interpretuje a zobrazí výsledek uživateli. Tím se webový prohlížeč, který je dostupný na všech počítačových platformách, stává univerzálním klientem pro libovolnou webovou aplikaci. [9]

2.2 Webový server

Webový server obsluhuje jednotlivé požadavky od webových prohlížečů uživatelů a sestavuje příslušné odpovědi, které odesílá zpět.

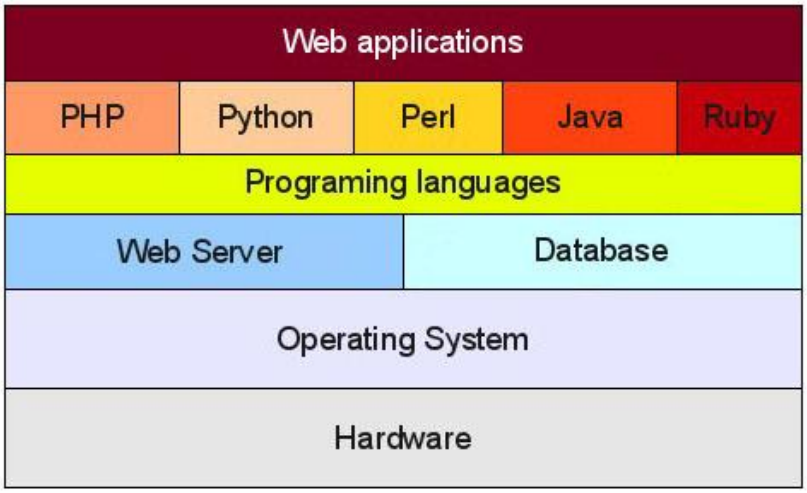
Obrázek 2.2a zobrazuje architekturu klient-server a způsob práce webového serveru. Uživatel nejdříve zvolí stránku, o kterou má zájem, zadá adresu do klienta (v našem případě do prohlížeče), ten odešle dotaz na webový server, server odpoví klientovi a nakonec je výsledek zobrazen uživateli.



Obr. 2.2a – Architektura klient-server a funkce webového serveru

Webový server může mít buď podobu samostatného počítače, nebo programu, tzv. démona, který běží na fyzickém hardwaru. [9] [10]

Obrázek 2.2b zobrazuje architekturu serveru při použití démona jako webového serveru.



Obr. 2.2b – Architektura serveru při použití démona

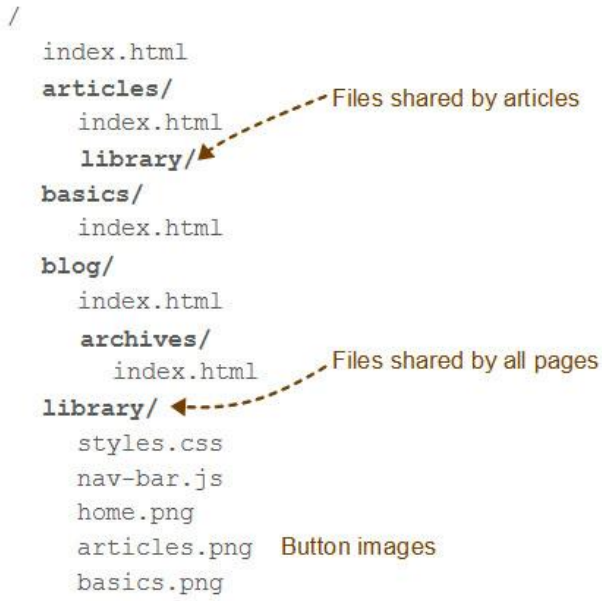
Na jednom webovém serveru může běžet několik různých webových aplikací, každá vytvořena v jiném programovacím jazyce.

Hardware webového serveru by měl být velmi rychlý, mít velkou kapacitu pro uložení dat a hodně paměti RAM potřebnou pro různé optimalizační techniky.

2.3 Struktura aplikace

Webová aplikace je uložena na speciálním místě na webovém serveru a skládá se z množství složek a souborů. Aplikace může obsahovat HTML stránky, PHP skripty, obrázky, konfigurační soubory a další soubory potřebné pro kompletní funkčnost aplikace.

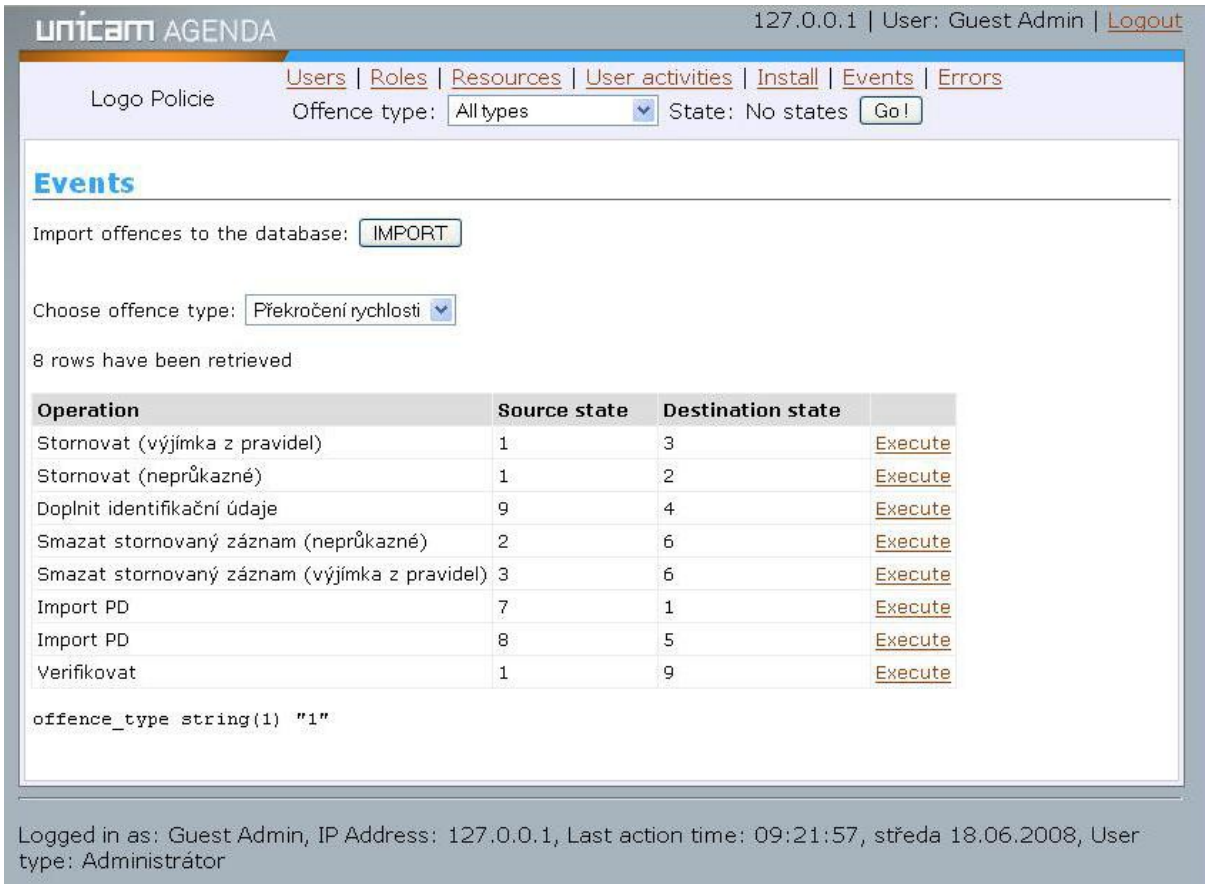
Obrázek 2.3a ukazuje příklad uspořádání souborů webové aplikace, soubory jsou rozděleny do příslušných složek dle jejich použití.



Obr. 2.3a – Příklad uspořádání souborů webové aplikace

Uživatel vnímá aplikaci jako kolekci webových stránek, pomocí kterých probíhá interakce. Stránky mohou obsahovat formuláře, tlačítka, různá menu, tabulky a mnoho dalších prvků.

Na obrázku 2.3b je příklad uživatelského rozhraní webové aplikace.



Obr. 2.3b – Uživatelské rozhraní webové aplikace

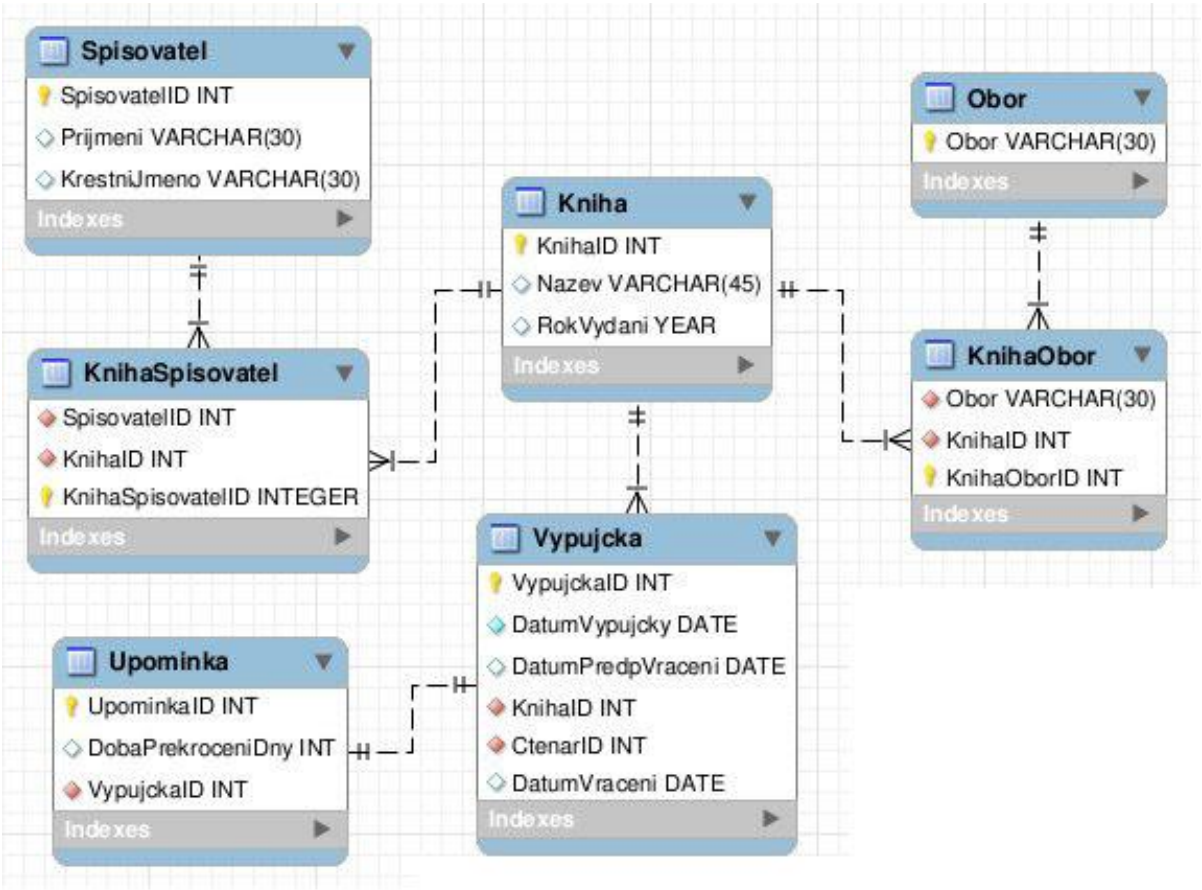
2.4 Databáze

Většina dnešních webových aplikací (jako elektronické obchody, diskuzní fóra, informační systémy škol, apod.) pracuje s dynamicky se měnícími daty, které je potřeba někde uchovávat, stejně jako rychle získávat. Pro tyto potřeby jsou využívány databáze uložené v databázovém serveru.

Databázový server má podobu programu, jenž poskytuje přístup k datům a provádí operace nad nimi.

Databáze je uspořádaná množina dat spolu s prvky určenými k efektivní manipulaci s nimi. Existuje několik databázových modelů, ty definují, jakým způsobem jsou data uložena a jaké mají mezi sebou vazby. Nejrozšířenějším modelem je relační model, kde jsou data ukládána v tabulkách a nad těmito tabulkami jsou definovány příslušné vazby.

Obrázek 2.4 zobrazuje databázový model aplikace – strukturu tabulek a jak jsou mezi sebou propojeny.



Obr. 2.4 – Databázový model aplikace

Návrh databázového modelu na začátku vývoje webové aplikace je velice důležitý, případné chyby vedou později k velikým problémům, popřípadě i ke zrušení celé aplikace. Při návrhu je třeba vytvořit tabulky pro všechny používané objekty, definovat všechny jejich parametry a určit, jaké budou mezi sebou mít vztahy.

Databáze po úvodním návrhu ve většině případů nesplňuje takové podmínky, že by mohla být okamžitě použita. Tudíž musíme databázi postupně normalizovat, tím eliminujeme takové problémy jako složené atributy záznamů nebo výskyt opakujících se záznamů v jedné tabulce.

Důležité je také rozhodnutí, která data se budou v databázi uchovávat. Čím více dat se ukládá, tím déle trvá vyhledávání potřebných dat, zvyšují se nároky na optimalizaci a je třeba více diskové kapacity. [1] [7]

3 Používané technologie

Kapitola 3 představuje používané technologie potřebné pro běh a správnou funkci webových aplikací. Technologie jsou v kapitolách 3.1 a 3.2 rozděleny na stranu serveru a stranu klienta.

Obecně lze říci, že dokument, který vrací webový server webovému prohlížeči jako odpověď, je vytvořen za pomoci několika různých programovacích jazyků. Logika a generování kompletního chování stránky probíhá na straně serveru, vizuální podoba a efekty na straně klienta.

3.1 Strana serveru

Pro správnou funkčnost aplikace musí běžet webový i databázový server a vše musí být naprogramováno bez chyb.

K vytváření webových aplikací se používají skupiny programů zvané „Solution stack“ obsahující webový server, databázový server a balík knihoven daného programovacího jazyka. Programy v dané skupině mají vestavěné funkce pro komunikaci mezi s sebou, proto je vývoj velmi rychlý a efektivní.

Nejznámější skupiny programů jsou LAMP (Linux, Apache, MySQL, PHP/Python/Perl) a WINS (Windows, IIS, .NET, SQL Server).

Pro tuto práci byla zvolena skupina programů AMP.

3.1.1 Apache

Apache HTTP Server je nejpoužívanější webový server, jedná se o softwarový webový server s otevřeným kódem a lze ho používat téměř na všech platformách. Svou oblíbenost si získal díky široké komunitě vývojářů, kteří jeho kvalitu ženou neustále kupředu, a rychlosti srovnatelnou s jinými papírově výkonnějšími webovými servery.

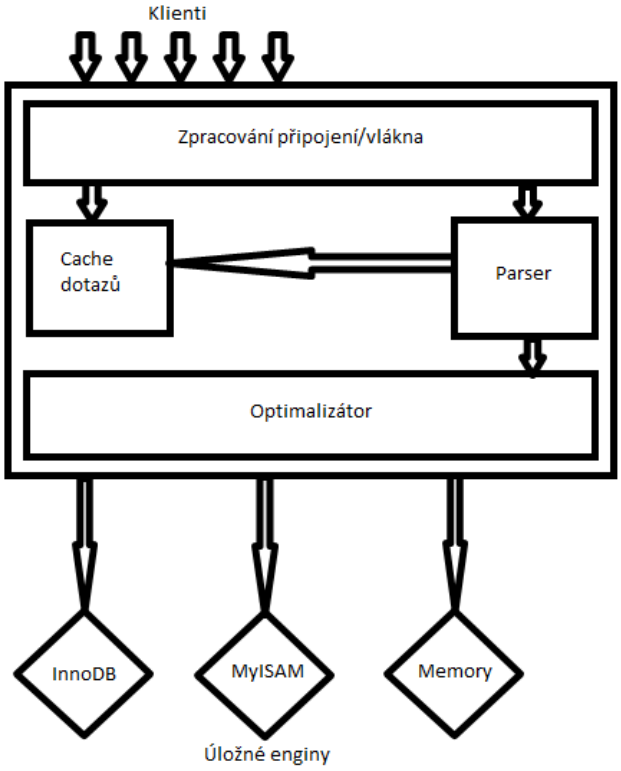
Apache také obsahuje moduly pro rozšíření funkcionality webové aplikace, například moduly pro autorizaci uživatelů, správu http hlaviček, mapování adresy URL na jinou URL a mnoho dalších.

3.1.2 MySQL

MySQL je rychlý a spolehlivý multiplatformní databázový systém. Vzhledem k jeho bezplatné licenci (je k dispozici i komerční) je využívám i mnoha mezinárodními firmami. Zmíněné výhody spolu se snadným použitím způsobily masové rozšíření. MySQL používá relační model dat.

Architektura MySQL je rozdělena na několik částí. Zahrnuje služby pro obsluhu klientů, část pro rozbor (parser), optimalizátor a úložné enginy, které zabezpečují ukládání a získávání dat uložených v MySQL. [1] [7]

Obrázek 3.1.2 ukazuje architekturu MySQL.



Obr. 3.1.2 – Architektura MySQL

3.1.3 PHP

Pro vytváření dynamických webových stránek se ve většině případů používá jednoduchý skriptovací jazyk s názvem PHP. Jedná se o jazyk interpretovaný, což znamená, že skripty v něm napsané se provádějí na straně serveru a zpět ke klientovi je poslána pouze vygenerovaná webová stránka.

Syntaxe jazyka PHP je podobná jazykům jako C, Java nebo Perl. Vlastnosti těchto jazyků kombinuje a vývojář má větší volnost. Kód se skládá převážně z příkazů, podmínek a funkcí. [2]

Obrázek 3.1.3 zobrazuje příklad skriptu napsaného v PHP.

```php
<?
$dbc=odbc_connect("gbook","","");
if (!$dbc)
{exit("Connection Failed: " . $dbc);}
$query="SELECT * FROM comments";
$rs=odbc_exec($dbc,$query);
if (!$rs)
  {exit("Error in SQL");}
echo '<h3>MS Access powered Guest Book</h3>';
while (odbc_fetch_row($rs))
{
  $e_name=odbc_result($rs,"name");
  $comment=odbc_result($rs,"comment");
  $e_date=odbc_result($rs,"entry_date");

 echo '<b>Name:</b>'.$e_name.'<br>';


echo '<b>Comments:</b> '.$comment.'<br /> <b>Date:</b> '.$e_date.'<br />
<hr/>';
}
odbc_close($dbc);
echo "</table>";

?>
```

Obr. 3.1.3 – Ukázka kódu PHP skriptu

Pro PHP existuje mnoho knihoven, např. pro spolupráci s již zmíněným MySQL, které z něj dělají mocný nástroj. Jazyk je neustále vyvíjen, podporuje více operačních systémů a nabízí svobodnou licenci, čímž se stal velmi populárním.

3.2 Strana klienta

Na straně klienta prohlížeč přijme dokument s generovaným zdrojovým kódem, sestaví výslednou webovou stránku a tu zobrazí uživateli.

Základní statické webové stránky obsahovaly pouze HTML kód, ovšem v současné době už téměř každá webová aplikace obsahuje dynamické webové stránky. Dynamické chování prvků na stránce je realizováno pomocí technologií jako Flash nebo Shockwave a většina prohlížečů už má integrovánu podporu nejrozšířenějšího Javascriptu.

3.2.1 HTML

HTML je jednoduchý značkovací jazyk pro vytváření webových stránek, na němž je postavený dnešní internet.

Jazyk má definovanou sadu značek, které popisují jednotlivé části dokumentu. Dokument napsaný v HTML má jasně předepsanou strukturu, která musí být dodržena.

Na obrázku 3.2.1 je HTML kód webové stránky.

```
<html>
    <div class="nadpis4">
      <h4>
        <u>Mé oblíbené stránky</u>
      </h4>
    </div>
    <ol type="1">
      <li><a href="http://www.seznam.cz">www.seznam.cz</a>
        <ul type="disc">
          <li>
            Text, který je zde napsán
          </li>
        </ul>
      </li>
    </ol>
    <p align="center">
      <a href="#p_zacatek">
        <u>Zpět na začátek</u>
      </a>
    </p>
</html>
```

Obr. 3.2.1 – HTML kód

Jazyk HTML je podporován na všech systémech, webové stránky lze proto zobrazit kdekoliv. [11]

3.2.2 CSS

K přesnějšímu popisu vzhledu webových stránek se využívá tzv. kaskádových stylů CSS. Ty se při vývoji zapisují mimo HTML kód a oddělí se tak obsah stránky od jejího vzhledu, což je velmi praktické, protože jednou napsané styly lze použít pro více stránek. Webový prohlížeč si poté tyto soubory se styly ukládá do mezipaměti, aby načítání v jednotlivých stránkách bylo rychlejší a nemusely se soubory pokaždé stahovat ze serveru.

Generovaná webová stránka v sobě obsahuje odkazy na jednotlivé soubory se styly, které se aplikují při zobrazení stránky. V některých případech jsou styly zapsány rovnou do kódu stránky.

Pomocí CSS lze nastavovat barvy pozadí, rozmísťovat elementy, nastavovat písma a mnoho dalšího. Syntaxe pro definici stylů je velmi jednoduchá a často obsahuje přímo i anglické výrazy pro popis některých vlastností.

Styly se nazývají „kaskádové“ kvůli respektování následujících pravidel:

- Zachovávají původní vzhled HTML elementů při změně pouze některé vlastnosti
- Podřízený element přejímá formátování nadřazeného elementu
- Při vícenásobné definici vlastnosti určitého stylu zůstává platný poslední zápis
- V případě konfliktu je brán v potaz element bližší formátovanému obsahu

Obrázek 3.2.2 zobrazuje příklad zápisu CSS stylů.

```
#mainmenu {
    /*text-align: right;*/
    float:left;
    width: 100%;
    margin: 0px 0px;
    border-top: 1px solid #727272;
    border-left: 1px solid #727272;
    border-right: 1px solid #727272;
    /*border-bottom: 1px solid #727272;*/
    background-color: #EEEEFE;
}

.login{
        float: left;
    width:300px;
    /*margin: 20px;*/
    text-align: right;
    /*height:150px;*/
    border: 1px solid #727272;
    background-color: #EEEEFE;
    /*padding: 5px 5px 5px 5px;*/
}

.incontent, .clientlogo {
        border: 1px solid #CCCCCC;
        margin: 3px 3px;
        padding: 5px 5px;
    background-color: #FFFFFF;
}
```

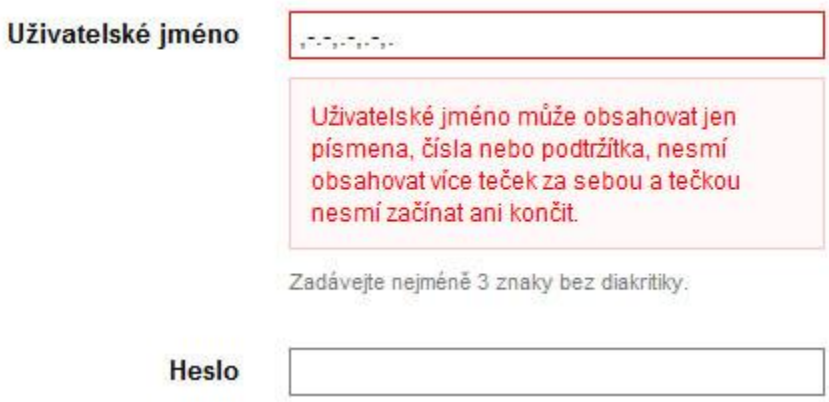
Obr. 3.2.2 – Zápis CSS stylů

Specifikace CSS se neustále rozšiřuje a vylepšuje. Proto jsou kaskádové styly brány jako standard pro tvorbu webových stránek. Drobnou nevýhodou může být pouze nejednotná podpora u všech webových prohlížečů, kdy některé prohlížeče interpretují určité vlastnosti rozdílně, což někdy vede ke špatné čitelnosti stránky. [12]

3.2.3 Javascript

Aby webové stránky byly skutečně interaktivní, využívá se skriptů napsaných v objektově orientovaném jazyce Javascript. Kódy skriptů jsou většinou vkládány přímo do webové stránky a spouští se až na straně klienta. Tímto způsobem je možno v reálném čase měnit vlastnosti elementů na stránce, ovládat je, či provádět různé efekty s obrázky. Skripty se rovněž používají k validaci vstupních dat formulářů nebo zpracování událostí kurzoru myši.

Obrázek 3.2.3 ukazuje příklad okamžité validace uživatelského jména pomocí Javascriptu.



Obr. 3.2.3 – Validace pomocí Javascriptu

Pomocí Javascript kódu nelze měnit data na serveru, lze pracovat pouze s lokálními zdroji. Jelikož se kód vykonává u klienta, zatěžuje jeho hardware, namísto toho je nesmírnou výhodou, že se kód interpretuje okamžitě a nemusí se čekat na odezvu serveru.

Javascript je multiplatformní a neustálé zvyšování rychlosti interpretů jeho používání drží na špici v žebříčku používání. [6]

3.3 Framework

Při vývoji aplikací se hojně využívá tzv. frameworků. Framework je kolekce knihoven poskytujících řadu podpůrných funkcí pro jednotlivé části aplikace. Obsahuje programovou logiku i často používané pomocné funkce a lze na něm postavit řešení vlastního problému.

Tyto knihovny jsou společné pro určité typy aplikací. Použitím frameworku se programátor vyhýbá nutnosti znovu implementovat stejnou funkcionalitu při vývoji dalších aplikací.

Využitím frameworku se sice celková rychlost aplikace zmenší a vývoj trvá déle, protože čas ušetřený použitím cizího kódu je třeba věnovat nastudování práce s frameworkem, ale pokud budou

na daném frameworku aplikace stavěny častěji, velké úvodní nastudování odpadá a vývoj se stane pohodlnější a efektivnější.

Pro webové aplikace se používají frameworky, které obsahují správu uživatelských relací, knihovny pro snadnou práci s databází nebo systém pro vytváření šablon.

Nejvíce populární návrhový vzor webových aplikačních frameworků je bezpochyby Model-View-Controler označovaný jako MVC. [4]

3.3.1 MVC architektura

Návrhový vzor Model-View-Controller (MVC) neboli Model-Pohled-Řadič usnadňuje vývoj a údržbu webových aplikací tím, že rozděluje datový model, uživatelské rozhraní a logiku aplikace do tří nezávislých částí. Programátoři pracující na aplikaci se tak mohou rozdělit do týmů, které pracují pouze na své části a všechny úpravy, které v dané části provedou, se nijak nepromítnout do zbylých dvou částí aplikace spravovaných ostatními týmy.

MVC architektura také předepisuje určitou adresářovou strukturu aplikace. Odděleny jsou jednotlivé soubory pro modely, pohledy i řadiče. Takto je výrazně zmenšena šance výskytu konfliktů v kódu a porovnání souborů s repozitářem verzí zabere podstatně kratší čas. [4]

MODEL

Model definuje logiku a data, se kterými aplikace pracuje. Zpracovává příkazy od řadiče a v případě potřeby reaguje na dotaz o stavu od pohledu. Obvykle obsahuje prostředky pro přístup k databázi, validaci, či autentifikaci.

VIEW

Pohled kompletuje změny provedené modelem do podoby vhodné k interakci. Představuje generovanou webovou stránku odeslanou uživateli. Obsahuje například již zmíněné technologie jako HTML nebo CSS.

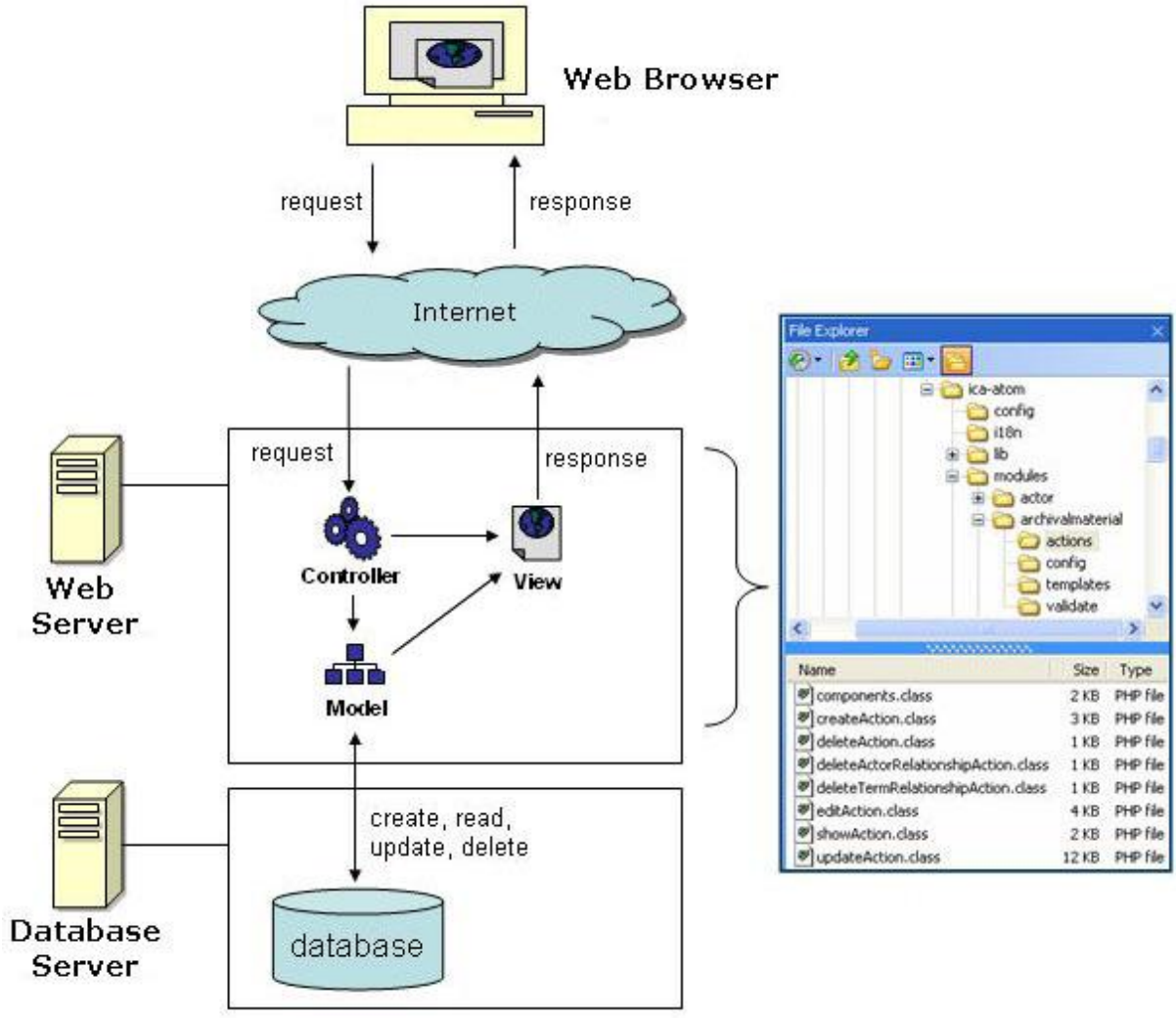
CONTROLLER

Řadič je mezivrstva, která vše spojuje, stojí mezi modelem a pohledem. Přijímá požadavky uživatele a podle nich posílá příkazy k příslušným změnám modelu, který je vykoná a poté jsou pohledem zobrazeny, nebo přímo pohledu.

Obecně celý proces probíhá takto:

- Uživatel provede akci v uživatelském rozhraní.
- Řadič dostane oznámení o akci.
- Řadič přistoupí k modelu či pohledu, který aktualizuje dle požadované akce.
- Pohled zobrazí aktualizovaná data použitím aktualizovaného modelu nebo příkazu od řadiče.
- Uživatelské rozhraní čeká na další akce uživatele, cyklus se opakuje.

Obrázek 3.3.1 zobrazuje MVC architekturu.



Obr. 3.3.1 – MVC architektura

Typickou implementací MVC návrhového vzoru postavenou na PHP a hojně používanou ve webových aplikacích je Zend Framework.

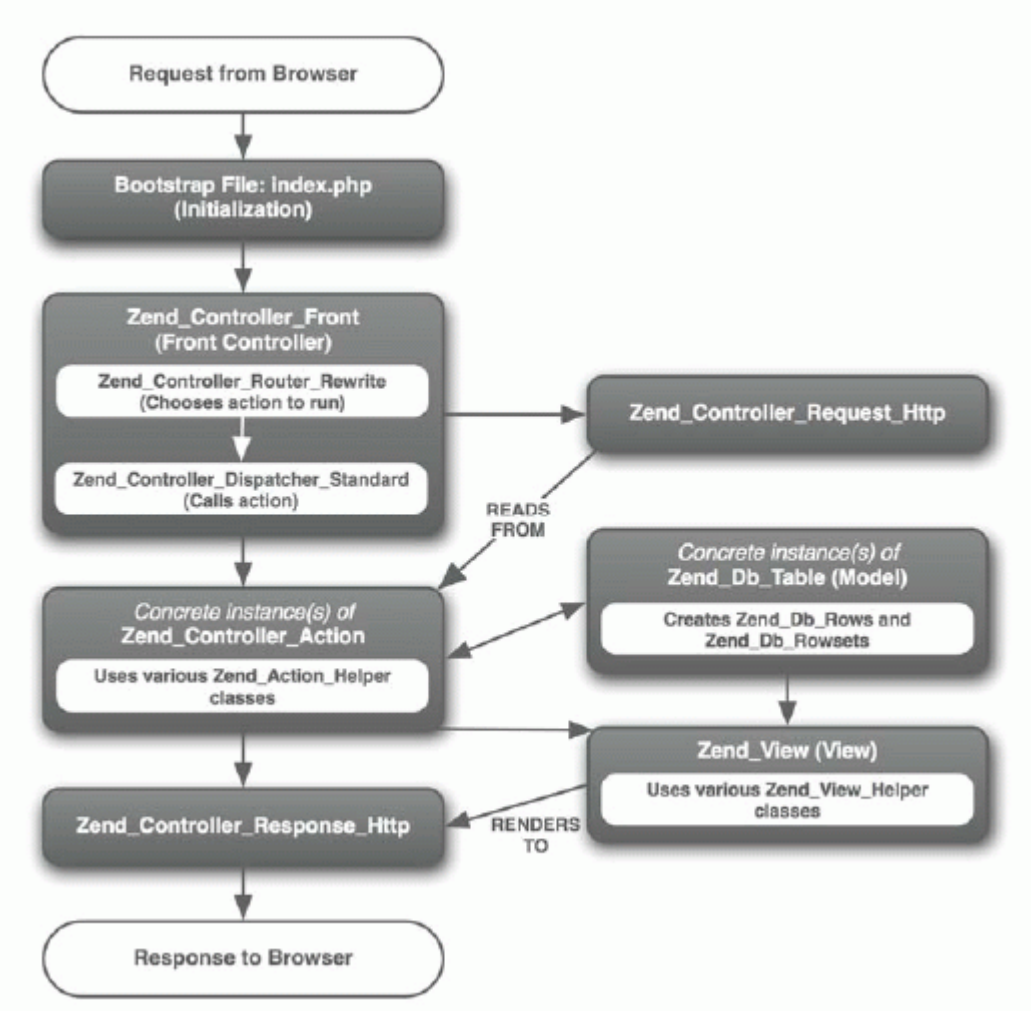
3.3.2 Zend Framework

Zend Framework byl vyvinut jako standardizovaný, objektově orientovaný, MVC framework pro PHP. Obsahuje velké množství komponent pro snadný vývoj většiny webových aplikací. Postavení aplikace na komponentách, které jsou širokou komunitou již otestované, ušetří obrovské množství času a výsledná aplikace se stane bezpečnější a spolehlivější.

Životní cyklus požadavku prohlížeče zpracovaným Zend Frameworkem se odvíjí od MVC architektury. Kvůli tomu musí být dodržena jak adresářová struktura aplikace, tak i pojmenování samotných objektů, jež se zpracování zúčastňují.

Na počátku je z webového serveru předán požadavek do Front Controlleru, o což se stará tzv. Bootstrap soubor, ve kterém jsou rovněž provedeny potřebné inicializace jako nastavení cest k Zend Frameworku samotnému nebo načtení potřebných tříd. Front Controller poté z dané URL v požadavku určí příslušný modul, kontroler a akci, kterou má provést. Samotný proces zpracování požadavku obstarává akce příslušného kontroleru, ve které už probíhá i přímá spolupráce s modely a pohledy. Jakmile je pohled kompletní, vrátí se jeho obsah jako odpověď klientovi. [4]

Obr. 3.3.2 ukazuje proces zpracování požadavku od klienta.



Obr. 3.3.2 – Zpracování požadavku v Zend Frameworku

Framework má otevřený zdrojový kód, který je před vydáním pečlivě testován, a tudíž ho může používat kdokoliv a zdarma. S každou novou verzí jsou do frameworku přidány další knihovny podporující novější technologie, proto lze aplikaci postavenou na nejaktuálnější verzi pokládat za moderní.

3.4 Analýza rychlosti

Pokud aplikaci postavíme na těchto základních technologiích, není zaručeno, že budou všechny funkce náležitě rychlé a že interakce s uživatelem bude vždy vyhovující.

Rychlost práce s webovými aplikacemi závisí na více faktorech. Velikost přenášeného obsahu, výkon hardwaru serveru i klienta, míra optimalizace databáze nebo propustnost sítě mohou tvořit slabá místa, které degradují výkon celé aplikace.

Z toho důvodu je třeba použít přídavné techniky eliminující jakékoliv možné zpomalení, které by zapříčinilo dlouhé čekání uživatele na odezvu.

Každá zrychlovací technika se ovšem nehodí na všechny typy webových aplikací, v této práci byly vybrány pouze některé, jejich popis a vlastnosti jsou uvedeny v kapitole 4, následně otestována odezva aplikace a zhodnoceno výsledné zrychlení v kapitole 5.

Aplikací několika technik zároveň lze eliminovat více slabých míst najednou a takto zvýšit výkon aplikace ještě více.

Změřena musí být odezva serveru zaznamenaná klientem, která je nejvíce směrodatná pro posouzení, zda výkon aplikace vyhovuje požadavkům.

4 Techniky zrychlení

Kapitola 4 představuje techniky vhodné k zrychlování webových aplikací.

Existuje obrovské množství technik, postupů a triků ke zvýšení rychlosti webových aplikací. Některé více a některé méně účinné.

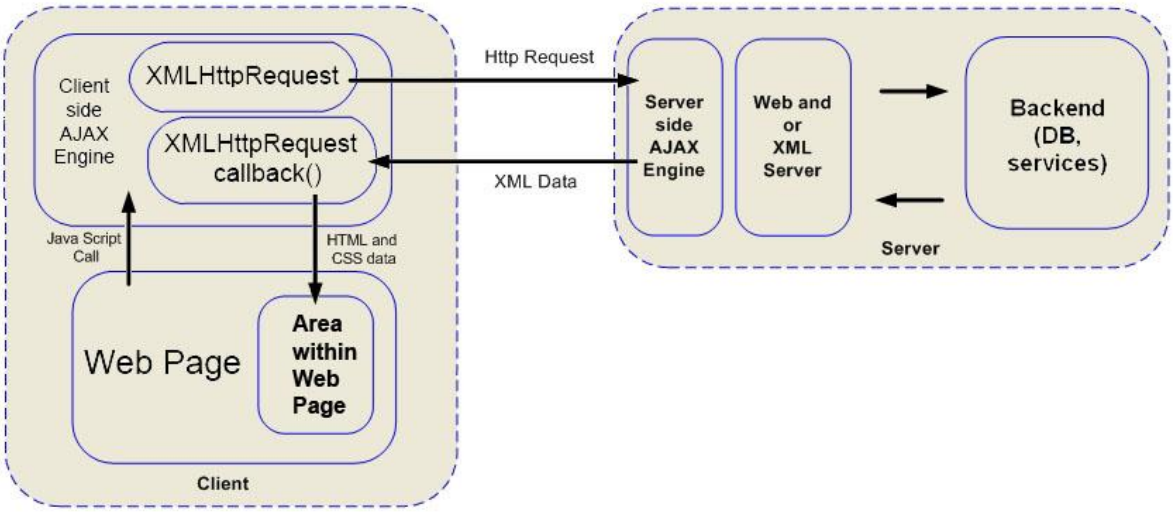
Zrychlovány by měly být především ty části aplikace, které v procesu zpracování požadavku zabírají nejvíce času. V kontextu moderních webových aplikací k tomuto účelu bezesporu slouží tyto aktivní technologie: AJAX, komprese, kešování výstupu a kešování dotazů a jejich výsledků, které jsou popsány v kapitolách 4.1, 4.2, 4.3 a 4.4.

4.1 AJAX

AJAX znamená „Asynchronnous Javascript And XML“, jedná se o techniku programování interaktivních webových stránek. Webová stránka je velká a složená z více částí. Pomocí AJAX lze zamezit znovuobnovení celé webové stránky při potřebě změny jejího obsahu. Obnoví se pouze požadovaná část.

Technika používá několik technologií, základem je odeslání požadavku na server na pozadí (asynchronně), což umožňuje XMLHttpRequest, pomocí něj i zpracujeme odpověď, PHP na serveru zpracuje požadavek a odešle zpět, požadavek je převeden a daná část stránky zobrazena pomocí HTML a CSS. Data mezi klientem a serverem jsou přenášena ve formě XML nebo JSON. Použití všech těchto technologií současně zajišťuje Javascript. [5]

Obrázek 4.1 zobrazuje zpracování požadavku pomocí AJAX.



Obr. 4.1 – Zpracování požadavku pomocí AJAX

XMLHttpRequest jako základní kámen AJAX-u může používat pro odesílání požadavků na server jak GET metodu, tak i POST. Pro jednoduché získání malého množství dat je vhodnější GET. GET žádosti bývají kešovány, což přispěje k dalšímu růstu výkonu při častém nahrávání stejných dat v krátkém časovém okamžiku. [6]

Použitím AJAX-u je množství přenášených dat výrazně redukováno, proto i čas mezi akcí uživatele a odezvou serveru je podstatně kratší. Server má mnohem méně práce se zpracováním požadavku, prohlížeč s vykreslením obsahu. Při dostatečně rychlém připojení se plynulost práce blíží desktopovým aplikacím.

Nevýhodou může být nutnost používání moderních webových prohlížečů, které podporují všechny potřebné technologie, to bývá problém u čím dál více se rozšiřujících mobilních zařízení. Vývojáři proto musí zajistit, aby aplikace byla dostupná i pro uživatele, kteří používají prohlížeč bez podpory technologie AJAX.

4.2 Komprese

Rychlost interakce webových aplikací závisí na rychlosti doručení vyřízených požadavků zpět ke klientovi. To přímo souvisí s rychlostí počítačové sítě.

Pokud by propustnost sítě omezovala rychlost komunikace mezi klientem a serverem, musely by být investovány další prostředky za účelem fyzického zvýšení propustnosti sítě. Tuto situaci lze ovšem elegantně vyřešit kompresí dat odesílaných ze serveru zpět ke klientovi. Jakmile jsou data komprimována, provoz na síti se sníží a klient zároveň pocítí, že aplikace reaguje rychleji.

Pro komprimaci webového obsahu se používají metody Gzip a Deflate. Gzip komprese je založena na Deflate algoritmu, ovšem na konec souboru ještě přidá pár extra informací, proto má soubor komprimovaný metodou Gzip vždy o něco málo větší velikost než Deflate. [13]

4.2.1 Komunikace

Na počátku komunikace musí webový prohlížeč oznámit serveru, že je schopný přijímat komprimované odpovědi. Tato informace se nachází v hlavičce požadavku jako parametr „Accept-Encoding“, pokud ji server zachytí, ví, že lze prohlížeči odesílat komprimované odpovědi. Pokud server vrací komprimovaný obsah, oznámí to prohlížeči v hlavičce odpovědi parametrem „Content-Encoding“.

Obrázek 4.2.1 ukazuje strukturu hlaviček při použití komprese.

Hlavičky odezvy zobrazit zdroj

Date	Thu, 05 May 2011 07:44:05 GMT
Server	Apache/2.2.17 (Win32) PHP/5.2.14
X-Powered-By	PHP/5.2.14
Expires	Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control	no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma	no-cache
Content-Encoding	gzip
Content-Length	484
Keep-Alive	timeout=5, max=100
Connection	Keep-Alive
Content-Type	text/html

Hlavičky požadavku zobrazit zdroj

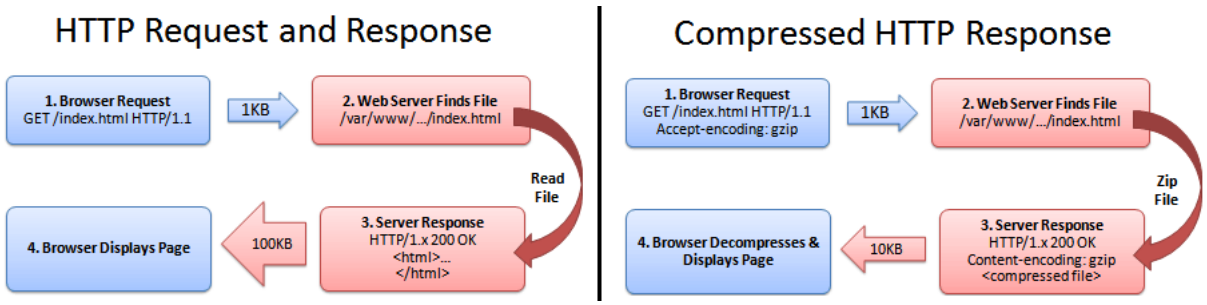
Host	localhost
User-Agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language	cs,en-us;q=0.7,en;q=0.3
Accept-Encoding	gzip, deflate
Accept-Charset	windows-1250,utf-8;q=0.7,*;q=0.7
Keep-Alive	115
Connection	keep-alive
Cookie	PHPSESSID=a3e1cb54949aa7129a7b2a5922676d9b

Obr. 4.2.1 – Struktura hlaviček při kompresi

4.2.2 Funkce

Pokud máme nastavenu kompresi odchozích dat v aplikaci na serveru, server nejdříve zjistí, které metody komprese prohlížeč podporuje. Když naše metoda není podporována, server neodešle v hlavičce parametr „Content-Encoding“ a je vrácena standardní nekomprimovaná odpověď. V opačném případě komprimuje cílový soubor a ten společně s informací o použité kompresi odešle zpět klientovi. Prohlížeč klienta zjistí, jakou metodou je obsah komprimován, dekomprimuje jej a výsledek zobrazí.

Obrázek 4.2.2 ukazuje porovnání generování odpovědi serverem standardně a při použití komprese.



Obr. 4.2.2 – Porovnání standardní a komprimované odpovědi serveru

Velikost odesílaných dat po kompresi bude podstatně menší než bez ní. Čas potřebný na kompresi a dekompresi souboru lze zanedbat. Komprese ovšem spotřebovává procesorový čas serveru i klienta, proto je třeba rozhodnout, zde má komprese smysl a opravdu urychlí běh aplikace. Toto rozhodnutí závisí na velikosti odpovědí serveru, propustnosti sítě a vzdálenosti klienta od serveru. V praxi je ale vhodné komprimovat všechny soubory větší 1kB. Výjimku tvoří obrázky a např. i PDF soubory, jelikož už jsou jednou komprimované a další komprese by akorát zbytečně vytěžovala procesor, nebo dokonce způsobila zvětšení velikosti souboru. [8]

4.2.3 Nastavení komprese

Starší verze Apache serveru používá Gzip kompresi a k nastavení jejich parametrů modul mod_gzip, novější verze Deflate kompresi a modul mod_deflate.

Pomocí těchto modulů lze serveru říci, které typy souborů se mají komprimovat, pro které prohlížeče má být zapnuta komprese, definovat velikost souborů pro kompresi, nebo nastavit stupeň komprese.

Obrázek 4.2.3 ukazuje aktivaci a nastavení komprese.

```
#aktivace modulu deflate
LoadModule deflate_module modules/mod_deflate.so

#pokud je modul aktivni, pouzij nasledujici nastaveni
<IfModule mod_deflate.c>
        #komprimuj pouze dane typy souboru
        AddOutputFilterByType DEFLATE text/plain
        AddOutputFilterByType DEFLATE text/html
        AddOutputFilterByType DEFLATE text/xml
        AddOutputFilterByType DEFLATE text/css
        AddOutputFilterByType DEFLATE application/xml
        #stupen komprese 9
        DeflateCompressionLevel 9
        #nastaveni omezeni komprese pro nektere prohlizece
        BrowserMatch ^Mozilla/4 gzip-only-text/html
        BrowserMatch ^Mozilla/4\.0[678] no-gzip
        BrowserMatch \bMSIE !no-gzip !gzip-only-text/html
</IfModule>
```

Obr. 4.2.3 – Nastavení parametrů komprese

Pokud by webový server přímo nepodporoval kompresi souborů, lze použít alternativu v podobě funkcí PHP *ob_start("ob_gzhandler")*, která se vloží na začátek každého souboru, jenž má být komprimován, a *ob_flush()*, která se vloží nakonec souboru. [14]

4.3 Kešování výstupu

Dnešní webové aplikace už se neskládají pouze ze statických HTML stránek a jednoduchých obrázků, kdy server vygeneroval soubor a poslal ho prohlížeči, který soubor přijmul a uložil v počítači uživatele. Při dalším požadavku na stejnou stránku se prohlížeč dotázal serveru, zda byla stránka změněna, pokud ne, zobrazil uloženou verzi stránky, jinak použil novou verzi.

S postupem času se webové aplikace stávají více a více komplexnější a dynamičtější, aby uspokojily potřeby náročných uživatelů. Proto je třeba hledat stále nové postupy a staré techniky odchází v zapomnění.

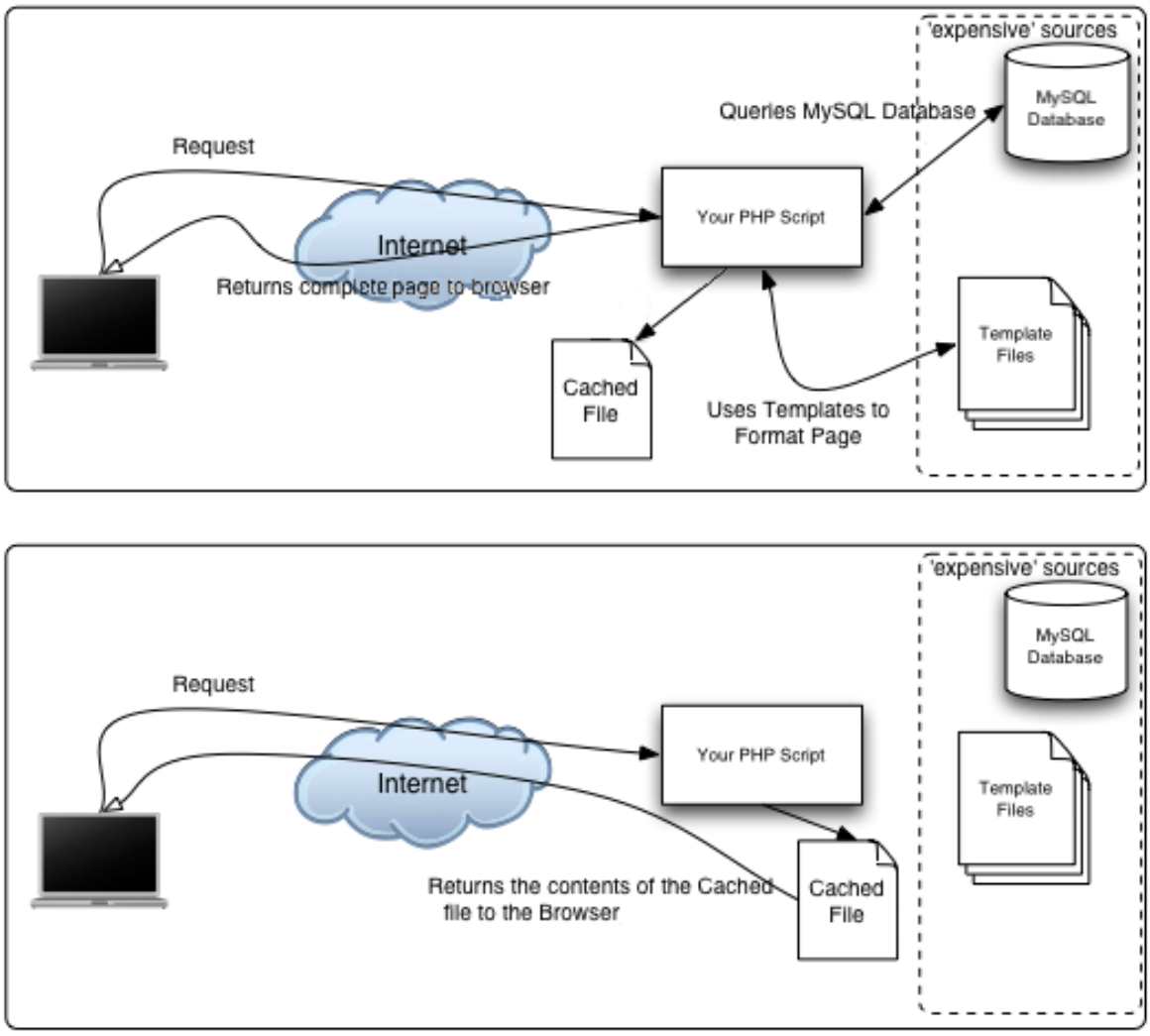
Celý proces zpracování požadavku na serveru způsobuje čekání uživatele na odpověď, pokud by stejný uživatel žádal o stejný obsah několikrát po sobě, server by pokaždé musel generovat danou stránku znovu. Stejná situace nastane, pokud o stejný obsah požádá několik uživatelů zároveň. Tuto situaci lze elegantně optimalizovat pomocí kešování výstupu na straně serveru. [15]

4.3.1 Funkce

Pokud uživatel odešle požadavek k získání určitého obsahu, musí server vykonat všechny potřebné úkony k vygenerování výsledné stránky, to zahrnuje získání dat z databáze, vložení dalších souborů nebo nejrůznější výpočty. Jedná-li se ovšem o požadavek, na který je stejná odpověď, jakou už uživatel dostal dříve, lze toho využít a ušetřit tak výkon serveru a čas na obdržení odpovědi klienta.

Při využití kešování se při prvním požadavku na daný obsah nejdříve uloží generovaný výsledek do souboru a následně odešle klientovi. Jakmile klient žádá téže obsah, server se podívá, zda byl už daný obsah generován a tudíž dříve uložen do souboru zastupující keš, a pokud ano, načte tento obsah a odešle klientovi jako výslednou odpověď.

Obrázek 4.3.1a zobrazuje použití kešovacího souboru.



Obr. 4.3.1a – Použití kešovacího souboru

K zachycení generovaného výsledku serverem se používají funkce PHP, které uchovávají data v paměti serveru.

Ukládání výstupu do paměti začne příkazem *ob_start*. Následně je všechen obsah, který by byl vypisován, ukládán. Po uložení zvoleného obsahu do paměti je výsledek načten pomocí funkce *ob_get_contents()* a uložen do souboru. Nakonec funkce *ob_end_flush()* ukončí použití paměti serveru a vypíše její obsah.

V obrázku 4.3.1b je jednoduchý příklad použití funkcí PHP pro kešování výstupu.

```php
<?php
//pokud existuje keesovaci soubor, nacti jeho obsah a zobraz ho
if (file_exists('cachefile.txt')) {
  readfile ( 'cachefile.txt' );
  exit();
}
//pokud soubor neexistuje, zapni ukladani vystupu do pameti
ob_start();
?>
//ukladany obsah
<html><body><h1>This page is a cached Page</h1></body></html>
<?php
//nacteni ulozeneho obsahu v pameti
$bufferContent = ob_get_contents();
//ulozeni obsahu do souboru
$fp = fopen('cachefile.txt', 'w');
fwrite($fp , $bufferContent);
fclose($fp);
//ukonceni pouzivani pameti a vypsani obsahu
ob_end_flush();
?>
```

Obr. 4.3.1b – Kešování výstupu

Jelikož se kešovaný obsah ve většině případů mění, musíme zajistit, abychom z kešovacího souboru načítali platná data. Na to existuje více způsobů, jak kontrolovat, zda kešovací soubor uchovává aktuální data a má-li už být obnoven. [16]

OBNOVENÍ KEŠOVACÍHO SOUBORU PO URČITÉM ČASE

Tento způsob se vyskytuje v nejvíce aplikacích. Podle toho, jak často se daná část aplikace mění, nastavíme dobu platnosti kešovacího souboru. Nejdříve zkontrolujeme, jestli je čas poslední modifikace souboru větší než aktuální čas mínus doba platnosti kešování. Pokud ano, obsah je platný a může být použit, v opačném případě se musí do paměti a následně do souboru uložit nový obsah, který je i následně vykreslen.

OBNOVENÍ KEŠOVACÍHO SOUBORU PO ZMĚNĚ OBSAHU

Po změně dynamického obsahu se nejdříve zkontroluje, zda se nový obsah rovná obsahu, který byl předtím uložen do kešovacího souboru, v kladném případě je vykreslen obsah souboru, jinak se novým obsahem přepíše starý, uložený v souboru a je rovněž vykreslen.

MANUÁLNÍ OBNOVENÍ KEŠOVACÍHO SOUBORU

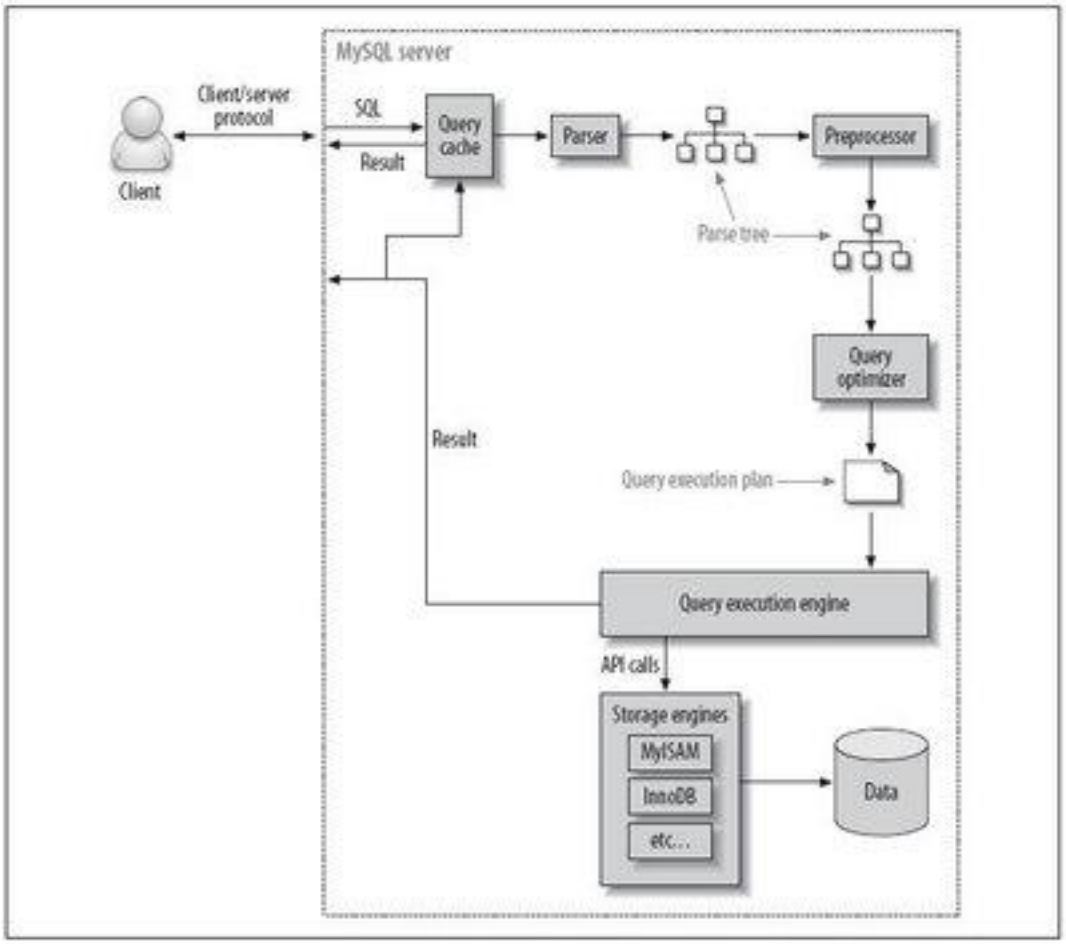
Obnovení kešovacího souboru probíhá na základě splněné podmínky. Tento způsob bývá většinou použit, pokud předchozí dva nejsou příliš vhodné.

4.4 Kešování dotazů a jejich výsledků

Dotazování na databázi je velmi častá elementární operace. Každý dotaz vrátí požadovaný výsledek ve formě seznamu záznamů, jejichž parametry splní dané podmínky.

Databázová keš MySQL serveru („MySQL query cache“) uchovává celé dotazy spolu s jejich výsledky. Pokud si klient vyžádá určité záznamy na základě dotazu, který už byl předtím proveden a tudíž je znám jeho výsledek, databázový server vrátí výsledek z keše, namísto toho, aby nový dotaz znovu analyzoval a vykonával.

Obrázek 4.4 ukazuje použití databázové keše.



Obr. 4.4 – Použití databázové keše

Databázové keše se využívají především v aplikacích, ve kterých se obsah příliš nemění a databázový server přijímá mnoho stejných dotazů.

Použití keše je vzhledem k aplikaci plně transparentní, aplikace neví, zda jsou data vrácena z keše nebo dotaz znovu vykonán.

4.4.1 Vlastnosti

V keši jsou uloženy kompletní výsledky SELECT příkazů. Rovněž se uchovávají informace o tom, s kterými tabulkami dané SELECT příkazy pracují. Jakmile se některá z tabulek změní, informace v keši přestanou být platné.

Kešovány nemohou být nedeterministické dotazy, tzn. dotazy, které obsahují funkce jako *NOW(), CURRENT_DATE()* nebo *CURRENT_USER()*, protože tyto funkce vrací výsledek závislý na času vykonání dotazu nebo přihlášeném uživateli a tudíž se výsledek dotazu mění. Dotazy, jež se odkazují na uživatelské proměnné, uživatelské funkce nebo dočasné tabulky také nemohou být kešovány. MySQL neví, zda dotaz obsahuje nedeterministickou funkci, dokud dotaz neanalyzuje. Pokud takovou funkci v dotazu najde, označí dotaz jako „nekešovatelný“ a neuloží k němu do keše žádný výsledek. Poté při příštím shodném dotazu zjistí, že v keši k němu není uložen patřičný výsledek a dotaz vykoná znova.

Samotné použití keše probíhá způsobem, že ještě než je vstupní dotaz analyzován, musí být prohledána keš. To s sebou ovšem nese režii navíc, stejně jako uložení dotazu do keše, pokud je označený jako „kešovatelný“ a ještě se v keši nenachází. Další režie nastává při zneplatnění obsahu keše, pokud se změní obsah tabulky, na kterou se kešovaný dotaz odkazuje. Režii těchto úkonů ale lze při vhodném použití a nastavení vzhledem k výslednému zrychlení zanedbat.

Způsobem, jakým můžeme určit, zda kešováním získáme nárůst výkonu, je zjištění počtu výsledků získaných z keše namísto výsledků získaných znovuprovedením dotazu. Toto číslo se určí pomocí vzorce:

$$\frac{\text{Qcache\_hits}}{(\text{Qcache\_hits} + \text{Com\_select})}$$

Kde *Qcache_hits* značí počet výsledků získaných z keše a *Com_select* počet výsledků získaných opětovným provedením dotazu.

Výsledné číslo udává procentuální úspěšnost keše. Úspěšnost keše závisí na typu aplikace a servírovaných dotazech, zpravidla už 30% bývá viditelný nárůst výkonu eliminující zmíněnou režii okolo. Pokud v keši nachází využití hodně složité dotazy, vyplatí se kešování i při nižší míře úspěšnosti.

V konečném důsledku nesmíme zapomenout, že kešování urychlí pouze proces získání výsledku dotazu, ne přenos výsledku ke klientovi. [19] [20]

5 Implementace a testování

Implementace aplikace proběhla s ohledem na potřeby dnešních aplikací. To zahrnuje dynamicky generovaný obsah, rozdělení na vrstvy, práce s externími soubory i práci s databází.

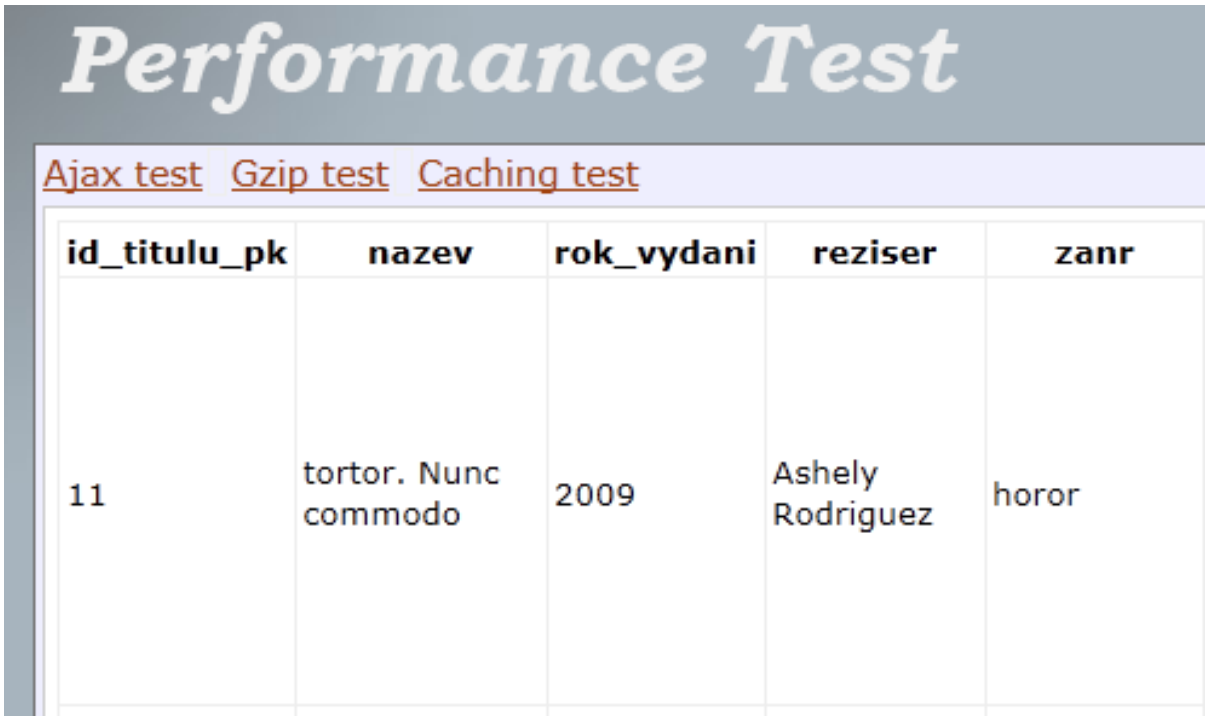
Aplikace byla nasazena na server s názvem Alenka, který běžel na operačním systému Debian 4.0 64bit a obsahující komponenty: procesor Intel Core2 6320 1,86GHz 64bit s 2 jádry, 4GB RAM a 2x500GB disky v SW RAID 1. Použitý webový server byl Apache 2.2.3, PHP verze 5.3.5 a MySQL verze 5.0.32.

Lokální testování se uskutečnilo na notebooku HP ProBook 4720s s procesorem Intel Core i3 M350 2,27GHz 64bit, 4GB RAM, 640GB pevným diskem s 5400ot./min. a operačním systémem Windows 7 Professional 64bit. Jako klientský prohlížeč byl použit Firefox 4.0.1.

5.1 Webová aplikace

Webová aplikace s názvem Performance byla vyvinuta jako příklad aplikace vhodná k účelnému testování daných technik zrychlení. Základ tvoří Zend Framework verze 1.10.8, programovací jazyk PHP a databáze MySQL. Také využívá funkce Javascriptu a formátování definované CSS styly.

Obrázek 5.1 ukazuje uživatelské rozhraní aplikace.



Obr. 5.1 – Uživatelské rozhraní aplikace

5.2 Nástroj měření

Pro měření výkonu webové aplikace se používají nástroje zachytávající odpovědi serveru. Nejdříve je třeba simulovat skutečnou zátěž serveru v praxi odesláním množství požadavků a poté změřit průměrný čas příjmu odpovědi.

K tomuto účelu byl použit nástroj JMeter.

5.2.1 JMeter

Jedná se o Java desktop aplikaci s otevřeným zdrojovým kódem k měření výkonu různých služeb. Zaměřuje se převážně na webové aplikace. Nástroj podporuje množství nastavení, testů, parametrizaci proměnnými, prezentaci výsledků za pomoci grafů nebo tabulek, odesílání různých požadavků a další operace.

Po spuštění JMeteru na klientském počítači je třeba nejdříve nastavit položku *Thread Group* v sekci *Test Plan*, ta definuje počet uživatelů, který má být simulován. Dále prodleva mezi požadavky jednotlivých uživatelů a počet požadavků pro každého uživatele.

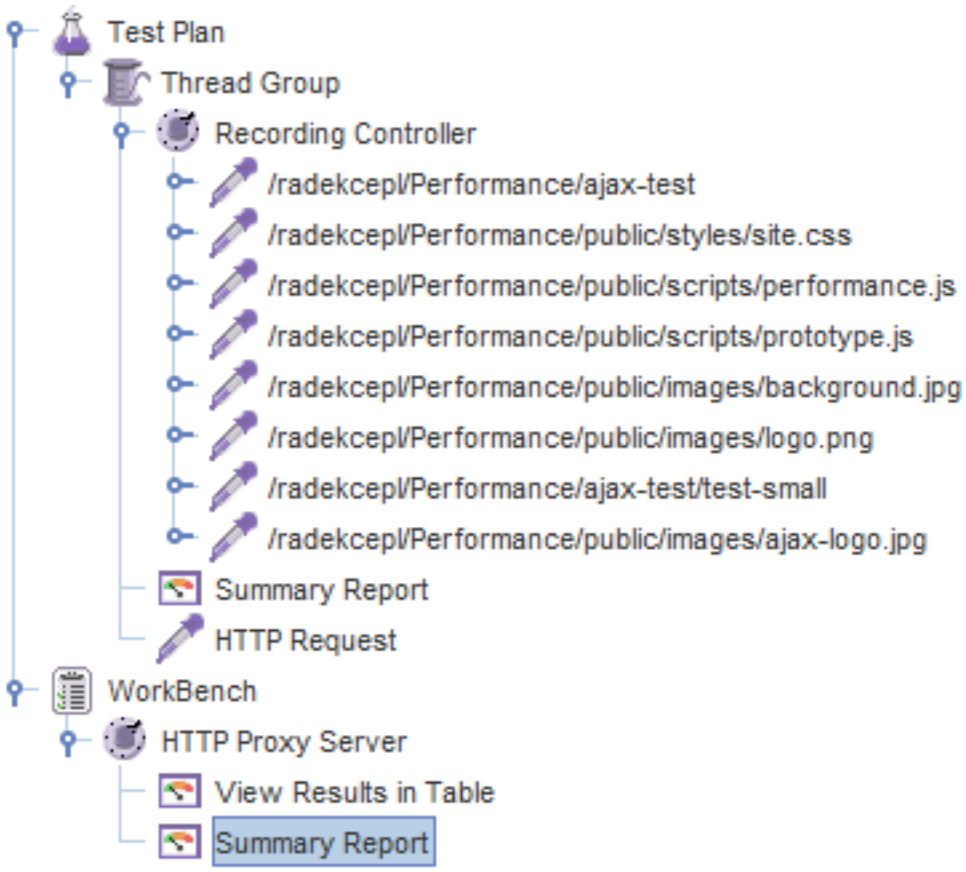
Další krok zahrnuje definici požadavku na server. Vytvoříme položku *HTTP Request*, u které zadáme jméno serveru nebo jeho IP adresu a poté cestu k souboru, který generuje měřenou webovou stránku.

Poslední krok se skládá z vytvoření „posluchače“, který bude zaznamenávat data při přenosu odpovědi a daným způsobem je zobrazí. Výsledná data můžou být zobrazena například do tabulky, v podobě grafu nebo stromu.

Aplikace Performance ovšem pracuje i s AJAX požadavky, které nelze takto generovat JMeterem a následně měřit odpovědi. Zde přišla chvíle pro nasazení proxy serveru, který musí být vytvořen v JMeteru a povolen ve Firefoxu.

Nejdříve je třeba vytvořit ovladač položkou *Record Controller*, do kterého se budou zaznamenávat všechny příchozí odpovědi. Následně proxy server položkou *HTTP Proxy Server* v sekci *Workbench*. Zde se nastaví port pro použití proxy serveru, všechny zachycené odpovědi se přesměrují do vytvořeného *Record Controlleru* a přidá se maska definující, pro který typ souborů se mají odpovědi zachytávat. Nakonec se přidají zvolení „posluchači“ k položce *HTTP Proxy Server* a vše pustí tlačítkem Start v *HTTP Proxy Server*. [17] [18]

V obrázku 5.2.1 je zobrazena struktura testovacího plánu v JMeteru pro aplikaci Performance.



Obr. 5.2.1 – Struktura testovacího plánu v JMeteru

Výsledky pro aplikované techniky zrychlení zachycené „posluchači“ JMeteru jsou použity a analyzovány v další kapitole.

5.3 Testy

V testech pro jednotlivé techniky jsou naměřeny skupiny hodnot reprezentující odezvy při získávání daného obsahu. Z těchto hodnot je patrná průměrná hodnota dané odezvy, výkyvy tak nejsou směrodatné. Čím menší odezva, tím rychleji proběhne získání daných dat a výkon výsledné aplikace se zvětší.

Testování proběhla pro několik velikosti obsahu, od nejmenšího reprezentovaného pouze obrázkem, přes střední s výsledky z databáze, po velký reprezentovaný kombinací všeho, od obrázků, přes ovládací prvky, dynamické prvky Javascriptu, až po výpis velkého množství dat z databáze.

5.3.1 AJAX

Pro implementaci AJAX technologie byl použit Javascriptový Framework s názvem Prototype, reprezentovaný knihovnou prototype.js.

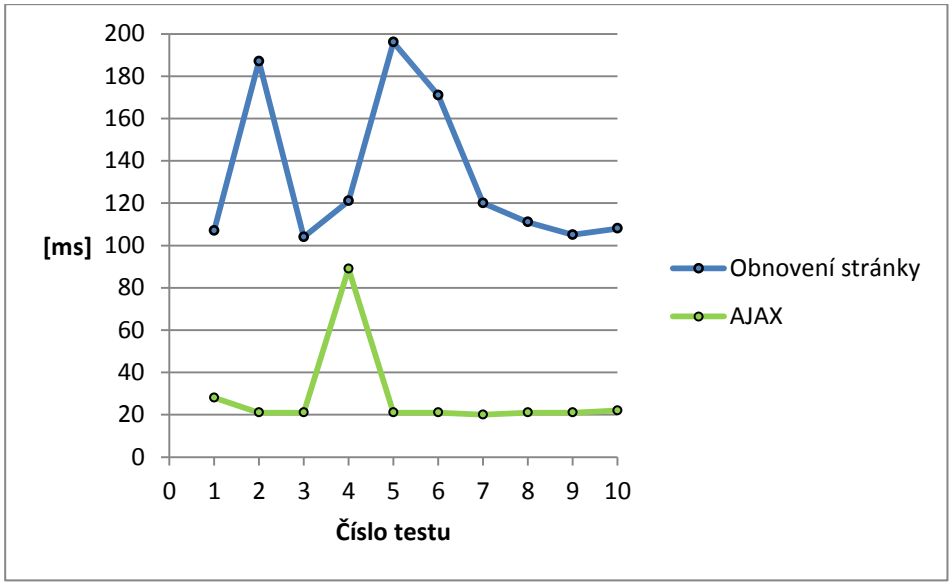
Odeslání XMLHttpRequestu bylo realizováno pomocí implementované Javascriptové funkce s názvem ajaxUpdater, která byla vyvolána jako akce na událost „onsubmit“ při stisku tlačítka typu „submit“ HTML formuláře.

MALÝ OBSAH

Doby trvání k vykreslení obsahu pro 10 testů:

MALÝ OBSAH	Čas [ms]										Průměr
Obnovení stránky	107	187	104	121	196	171	120	111	105	108	**133**
AJAX	28	21	21	89	21	21	20	21	21	22	**28,5**

Průměrné zrychlení: **78,6%**



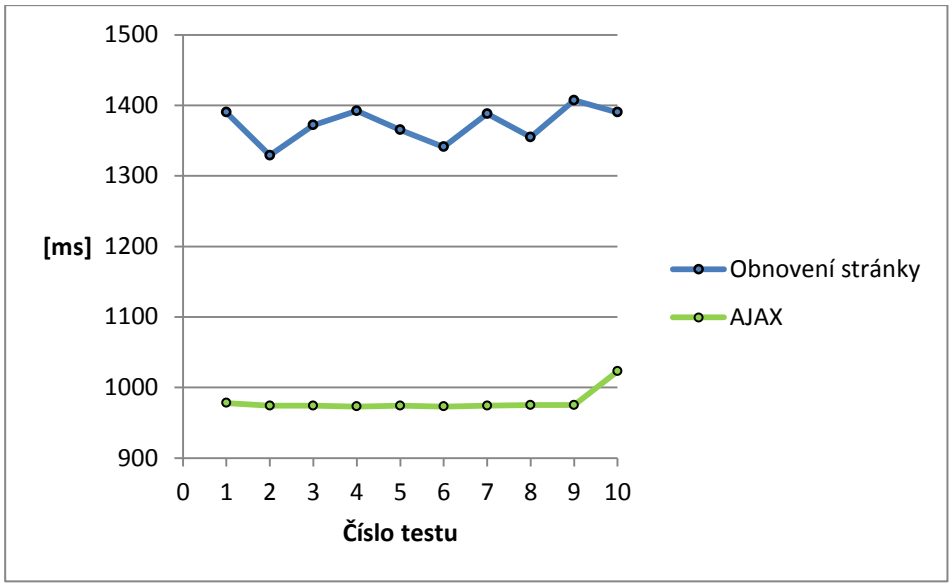
Obr. 5.3.1a – Výsledky pro malý obsah

STŘEDNÍ OBSAH

Doby trvání k vykreslení obsahu pro 10 testů:

STŘEDNÍ OBSAH	Čas [ms]										Průměr
Obnovení stránky	1390	1329	1372	1392	1365	1341	1388	1355	1407	1390	**1372,9**
AJAX	978	974	974	973	974	973	974	975	975	1023	**979,3**

Průměrné zrychlení: **28,7%**



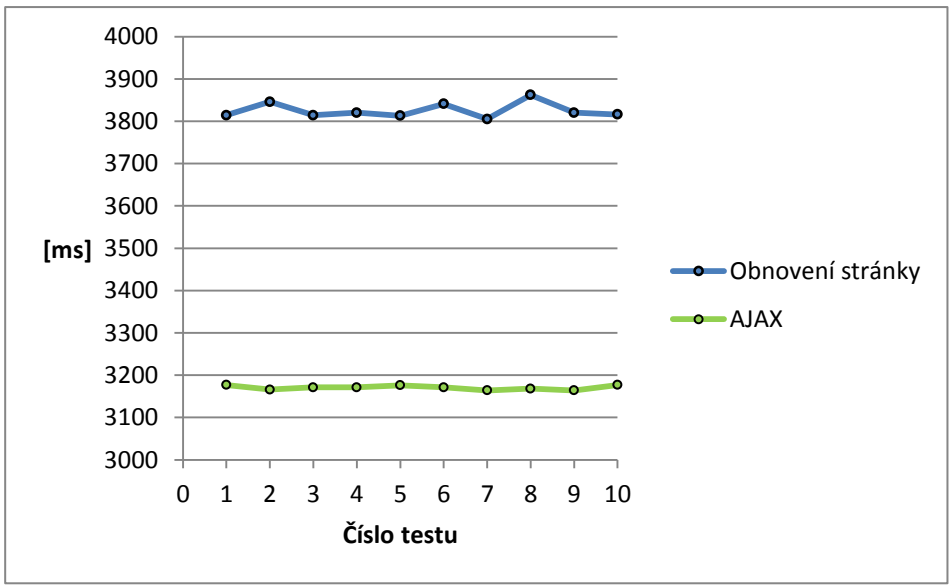
Obr. 5.3.1b – Výsledky pro střední obsah

VELKÝ OBSAH

Doby trvání k vykreslení obsahu pro 10 testů:

VELKÝ OBSAH	Čas [ms]										Průměr
Obnovení stránky	3814	3846	3814	3820	3813	3841	3805	3862	3820	3816	**3825,1**
AJAX	3177	3166	3171	3171	3176	3171	3164	3168	3164	3177	**3170,5**

Průměrné zrychlení: **17,1%**



Obr. 5.3.1c – Výsledky pro velký obsah

Pro AJAX technologii byl zjištěn nárůst výkonu od **17,1%** do **78,6%** podle velikosti požadovaného obsahu. Toto zrychlení je závislé na poměru velikosti daného obsahu k velikosti zbytku obsahu stránky. AJAX se tak stává nejvýhodnější při nahrávání menšího obsahu v rozsáhlých strukturovaných aplikacích.

5.3.2 Komprese

Pro komprimaci byl použit DEFLATE mód.

Přičemž filtr pro typy souborů vypadal následovně:

<FilesMatch "\\.(js|css|html|htm|php|xml)$">

SetOutputFilter DEFLATE

</FilesMatch>

Komprese tedy zahrnovala všechny použité soubory vyjma obrázků, které nebylo třeba komprimovat.

Výsledné velikosti a nahrávací časy souborů:

Soubor	KOMPRESE VYPNUTÁ		KOMPRESE ZAPNUTÁ	
	Velikost [kB]	Nahrávací čas [ms]	Velikost [kB]	Nahrávací čas [ms]
gzip-test	235,6	3791	38,6	3634
site.css	4,5	3	1,3	2
performance.js	0,5	10	0,2	9
prototype.js	123,2	34	28,5	23
background.jpg	10,4	8	10,4	8
logo.png	3	19	3	19
Celkem	**377,2**	**3865**	**82**	**3695**

Z výsledků je patrné, že komprimace výrazně zmenšila velikosti všech zvolených souborů a to z 377,2kB na 82kB, tedy celkově o **78%**, což je podstatný úbytek, který i v tomto případě významně ušetří zatížení sítě. Rovněž se o několik procent zrychlilo nahrávání celé stránky.

Pokud by se aplikace skládala z desítek souborů o velikosti několik stovek kB a více, bylo zatížení sítě daleko více redukováno a v tomto případě už by bylo nasazení komprese nutné.

5.3.3 Kešování výstupu

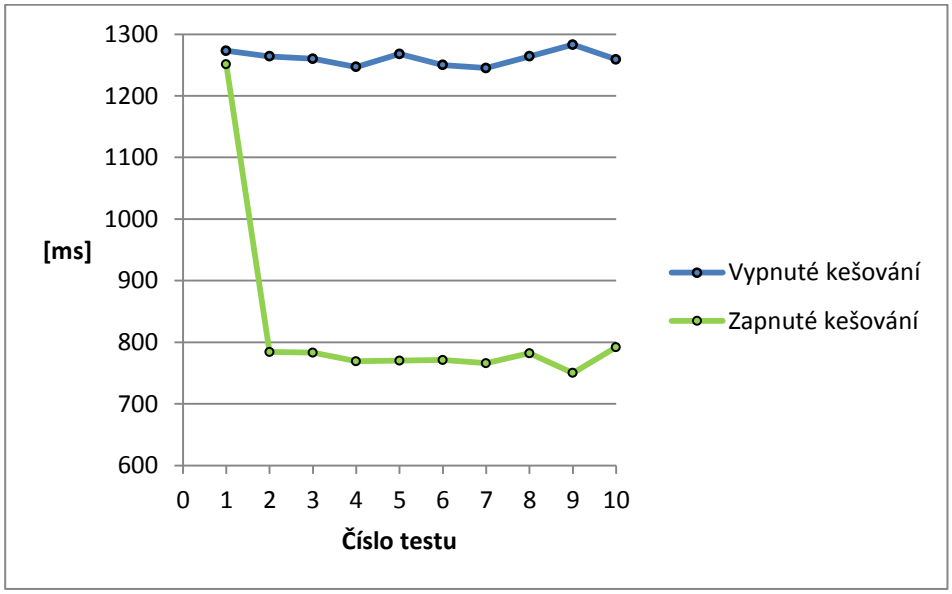
Kešování výstupu probíhalo do textového souboru. Testy byly realizovány pro 2 velikosti obsahu.

MENŠÍ OBSAH

Doby trvání k vykreslení obsahu pro 10 testů:

MENŠÍ OBSAH	Čas [ms]										Průměr
Vypnuté kešování	1273	1264	1260	1247	1268	1250	1245	1264	1283	1259	**1261,3**
Zapnuté kešování	1251	784	783	769	770	771	766	782	750	792	**821,8**

Průměrné zrychlení: **34,8%**



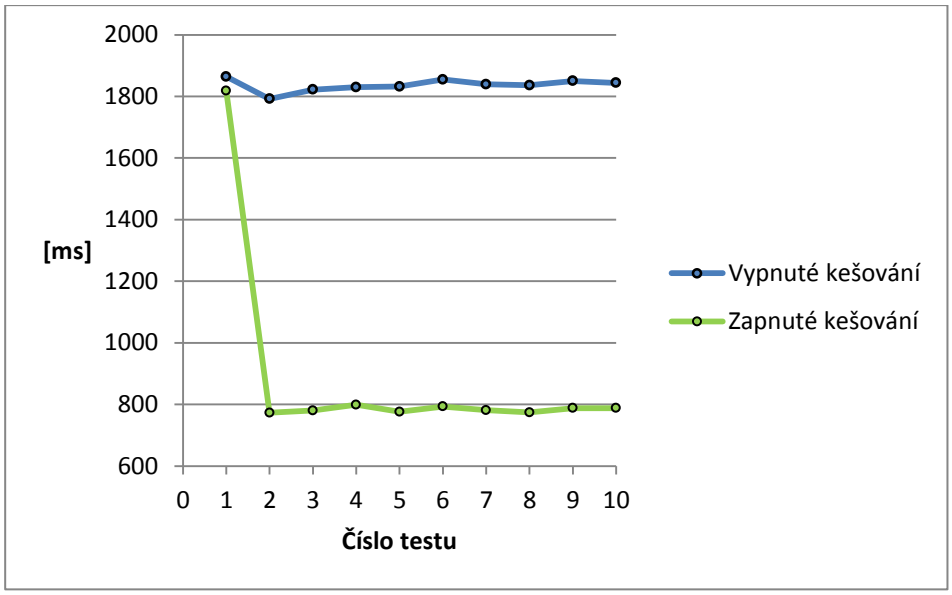
Obr. 5.3.3a – Výsledky pro menší obsah

VETŠÍ OBSAH

Doby trvání k vykreslení obsahu pro 10 testů:

VETŠÍ OBSAH	Čas [ms]										Průměr
Vypnuté kešování	1864	1792	1822	1830	1832	1855	1839	1836	1850	1844	**1836,4**
Zapnuté kešování	1818	773	780	799	776	793	781	774	788	788	**887**

Průměrné zrychlení: **51,7%**



Obr. 5.3.3b – Výsledky pro větší obsah

V testech byla dosažena zrychlení **34,8%** a **51,7%**. Tato čísla ukazují, že kešování výstupu je velmi důležitá technika a její síla roste s větším množstvím uloženého obsahu, který by byl potřeba znovu celý vygenerovat. Technika ovšem najde širší uplatnění spíše pro menší obsah, jako různá menu apod., u kterých se nemusí tolik řešit obnovování kešovacího souboru.

5.3.4 Kešování dotazů a jejich výsledků

Vzhledem k velikosti testovací aplikace a databáze nebylo možné naplno sledovat zvýšení výkonu při použití databázové keše změřením času doručení požadavků.

Skripty se generují několik stovek milisekund na rozdíl od provádění dotazů, které trvá jednotky, maximálně desítky milisekund pro výpis několika stovek až tisíc záznamů.

Kešování dotazů a jejich výsledků by se výrazněji projevilo pro větší databáze s rozsahem několika desítek tabulek a statisíci záznamy.

Proto bylo testování prováděno na úrovni databáze. Zde se dané výhody kešování projevily dostatečně.

Měření je rozděleno na 3 části dle náročnosti dotazu, tu udává počet zúčastněných tabulek, složitost podmínky a počet vrácených záznamů.

MALÝ DOTAZ

Dotaz:

use performance;

select zak.jmeno as jmeno_zakaznika, zak.prijmeni as prijmeni_zakaznika, sum(tit.cena) as celk_cena

from zakaznik as zak

join vypujcka as vyp on zak.id_zakaznika_pk = vyp.id_zakaznika_fk

join titul as tit on vyp.id_titulu_fk = tit.id_titulu_pk

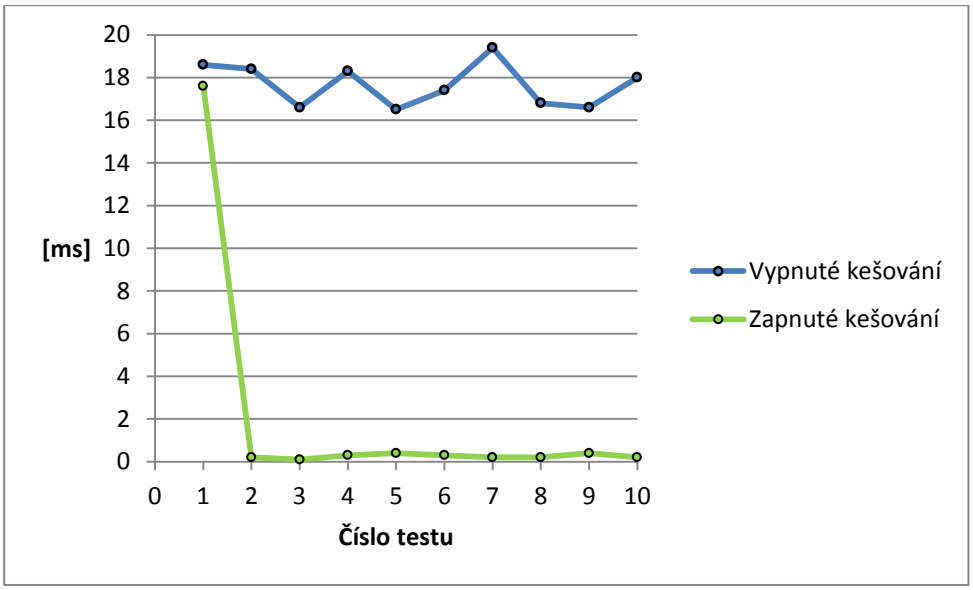
group by zak.id_zakaznika_pk;

Počet výsledků:

50

Doby trvání vykonání dotazu pro 10 testů:

MALÝ DOTAZ	Čas [ms]									
Vypnuté kešování	18,6	18,4	16,6	18,3	16,5	17,4	19,4	16,8	16,6	18
Zapnuté kešování	17,6	0,2	0,1	0,3	0,4	0,3	0,2	0,2	0,4	0,2



Obr. 5.3.4a – Výsledky pro malý dotaz

STŘEDNÍ DOTAZ

Dotaz:

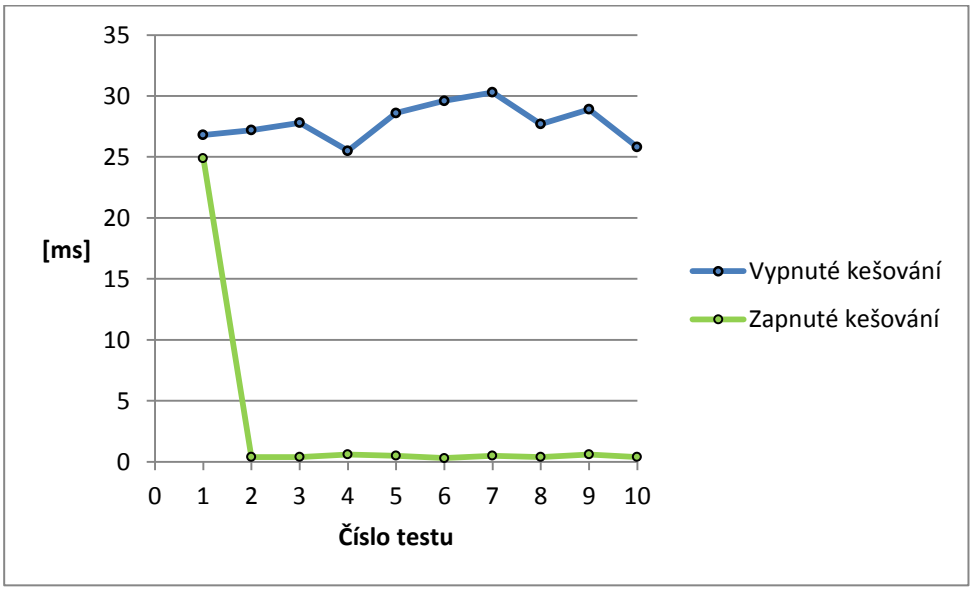
```
use performance;
select vyp.datum_vypujceni, vyp.datum_vraceni, vyp.stav, tit.nazev, tit.reziser, tit.cena
from vypujcka as vyp
join titul as tit on vyp.id_titulu_fk = tit.id_titulu_pk
where tit.cena BETWEEN 110 AND 120
order by vyp.datum_vypujceni;
```

Počet výsledků:

559

Doby trvání vykonání dotazu pro 10 testů:

STŘEDNÍ DOTAZ	Čas [ms]									
Vypnuté kešování	26,8	27,2	27,8	25,5	28,6	29,6	30,3	27,7	28,9	25,8
Zapnuté kešování	24,9	0,4	0,4	0,6	0,5	0,3	0,5	0,4	0,6	0,4



Obr. 5.3.4b – Výsledky pro střední dotaz

VELKÝ DOTAZ

Dotaz:

use performance;

select vyp.datum_vypujceni, vyp.datum_vraceni, vyp.stav, zak.jmeno as jmeno_zakaznika,

zak.prijmeni as prijmeni_zakaznika, tit.nazev, tit.reziser, tit.cena, zam.jmeno as jmeno_zamestnance,

zam.prijmeni as prijmeni_zamestnance

from vypujcka as vyp

join titul as tit on vyp.id_titulu_fk = tit.id_titulu_pk

join zamestnanec as zam on vyp.id_zamestnance_fk = zam.id_zamestnance_pk

join zakaznik as zak on vyp.id_zakaznika_fk = zak.id_zakaznika_pk

where tit.cena BETWEEN 70 AND 145

and (zam.prijmeni like '%a%' or zam.jmeno like '%e%')

and (zak.jmeno like '%a%')

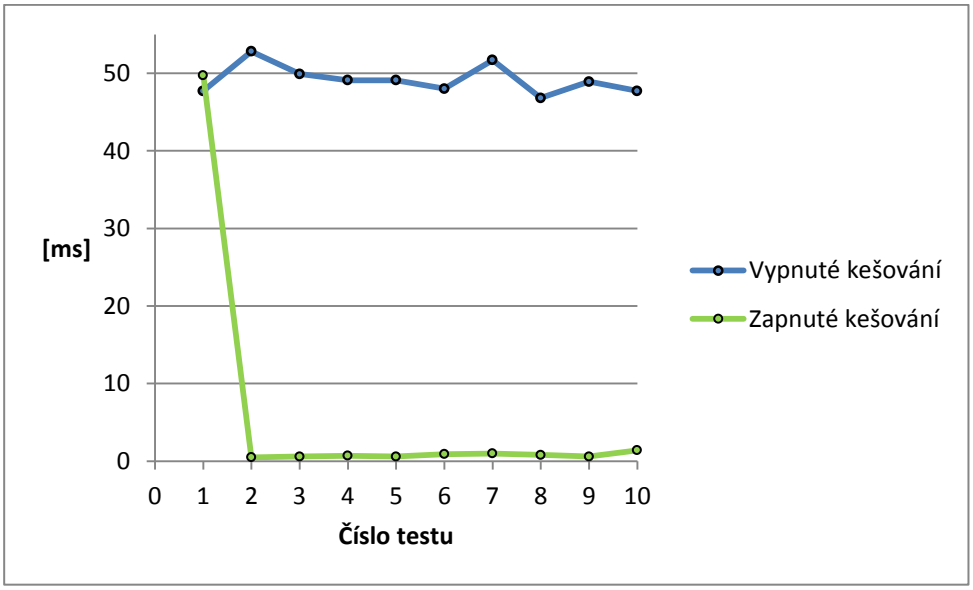
order by tit.nazev, tit.reziser;

Počet výsledků:

2090

Doby trvání vykonání dotazu pro 10 testů:

VELKÝ DOTAZ	Čas [ms]									
Vypnuté kešování	47,7	52,8	49,9	49,1	49,1	48	51,7	46,8	48,9	47,7
Zapnuté kešování	49,7	0,5	0,6	0,7	0,6	0,9	1	0,8	0,6	1,4



Obr. 5.3.4c – Výsledky pro velký dotaz

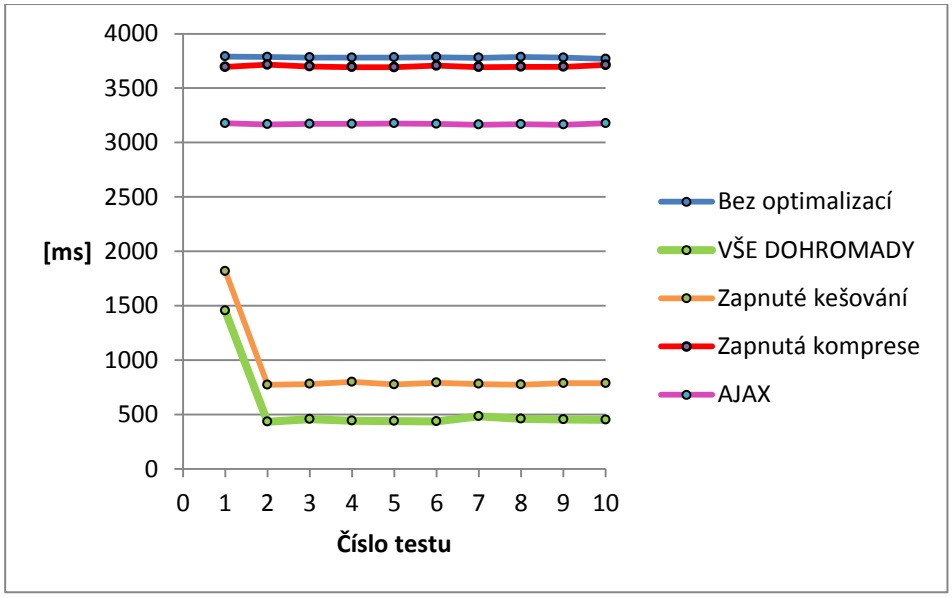
Jak ukazují výsledky, po prvním uložení dotazu a výsledku do keše je každé další zpracování rychlejší zhruba 100x, což znamená obrovský přínos v podobě menší zátěže databázového serveru a i rychlejšího vrácení požadovaných výsledků z databáze uživateli.

5.3.5 AJAX + Komprese + Kešování výstupu

V závěrečném testu byly postupně aplikovány metody AJAX, kešování a komprese pro získání velkého obsahu a nakonec aktivovány všechny tyto techniky zároveň a získán nejvyšší stupeň optimalizace.

Doby trvání získání obsahu pro 10 testů:

VELKÝ OBSAH	Čas [ms]										Průměr
Bez optimalizací	3791	3786	3782	3780	3781	3785	3779	3786	3780	3768	**3781,8**
Zapnuté kešování	1818	773	780	799	776	793	781	774	788	788	**887**
Zapnutá komprese	3695	3716	3699	3694	3692	3706	3693	3697	3697	3713	**3700,2**
AJAX	3177	3166	3171	3171	3176	3171	3164	3168	3164	3177	**3170,5**
VŠE DOHROMADY	1456	435	457	443	440	436	485	461	455	453	**552,1**



Obr. 5.3.5 – Kompletní výsledky pro velký obsah

Použitím všech technik zároveň bylo dosaženo úctyhodného zrychlení **85,4%**. Takovéto zrychlení odezvy serveru znamená pro uživatele rapidně vyšší rychlost práce s aplikací a požadovaný obsah získá téměř okamžitě, aniž by musel několik sekund čekat. Navíc velikost výsledného obsahu se díky kompresi zmenšila z **199,7kB** na **42,4kB**, tj. o **78,8%**, tím pádem se výrazně snížilo zatížení sítě.

6 Závěr

Diplomová práce se zabývala problematikou zrychlování moderních webových aplikací. Tohoto cíle bylo dosaženo.

Nejdříve jsem prostudoval literaturu zabývající se návrhem webových aplikací s ohledem na jejich maximální rychlost. Znalosti posloužily k představení funkce a struktury moderních webových aplikací současnosti v první části práce.

Následně jsem se věnoval analýze možností zvyšování rychlosti webových aplikací. Byla odhalena slabá místa a vybrány techniky k jejich odstranění.

Testovací moderní webová aplikace byla implementována a následně otestována za použití vhodných svobodných technologií a platforem.

Ke zrychlení testovací webové aplikace posloužilo postupné a následně společné použití vybraných technik a poté změřeno zvýšení rychlosti.

Jednotlivými technikami se podařilo získávat zrychlení v řádu několika desítek procent, při společném nasazení vybraných technik bylo dosaženo zrychlení až 85%.

Takovýmto spojením optimalizačních technik lze aplikaci zrychlit na úroveň, která smazává potřebu jakkoliv zrychlovat hardware. Výsledky testování navíc vypovídají o skutečnosti, že dané techniky byly zvoleny a použity správně.

Samozřejmě toto není obecný postup zrychlení všech webových aplikací, v případě použití většího množství generovaného obsahu by bylo vhodné použít jiné techniky, taktéž u aplikací obsluhující miliony uživatelů je třeba zvolit jiné postupy pro zvýšení výkonu.

Další možná práce na toto téma by zahrnovala použití a otestování ostatních zrychlovacích technik, jejichž nasazení spolu se mnou testovanými technikami by mohlo vést ještě k většímu zrychlení. Testování by rovněž mohlo být aplikováno na další typy obsahu vyskytujících se v dnešních webových aplikacích.

Literatura

[1] Brad Bulger, Jay Greenspan and David Wall: *MySQL/PHP Database Applications, Second Edition*. Wiley Publishing, Inc., Indianapolis, 2004. ISBN 0-7645-4963-4

[2] Marc Wandschneider: *Core Web Application Development with PHP and MySQL*. Addison Wesley Professional, září 2005. ISBN 0-13-186716-4

[3] Kolektiv autorů: *Database Design*. Elsevier Inc., Burlington, 2009. ISBN 978-0-12-374630-6

[4] Kolektiv autorů: *Pro PHP: Patterns, Frameworks, Testing and More*. Springer-Verlag New York Inc., 2008. ISBN 978-1-4302-0279-0

[5] Steve Souders: *Even Faster Web Sites*. O'Reilly Media, Inc., 2009. ISBN 978-0-596-52230-8

[6] Nicolas C. Zakas: High Performance Javascript. O'Reilly Media, Inc., 2010. ISBN 978-0-596-80279-0

[7] Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, Jeremy D. Zawodny, Arjen Lentz, and Derek J. Balling: High Performance MySQL, Second Edition. O'Reilly Media, Inc., 2008. ISBN 978-0-596-10171-8

[8] Steve Souders: High Performance Web Sites. O'Reilly Media, Inc., 2007. ISBN 978-0-596-52930-7

[9] Petr Palas: Historie Webu z pohledu vývojáře [online]. Brno, duben 2001. Dostupné na URL: http://www.fi.muni.cz/usr/jkucera/pv109/2001/xpalas.htm

[10] DocForge, An Open Wiki For Software Developers: Web Application [online]. Dostupné na URL: http://docforge.com/wiki/Web_application

[11] WEBTVORBA: Přehled webových technologií [online]. Leden 2009. Dostupné na URL: http://webtvorba.howto.cz/prehled-webovych-technologii

[12] Dušan Janovský: CSS kurz [online]. Poslední aktualizace v listopadu 2010. Dostupné na URL: http://www.jakpsatweb.cz/css

[13] BetterExplained: How To Optimize Your Site With GZIP Compression [online]. Dostupné na URL: http://betterexplained.com/articles/how-to-optimize-your-site-with-gzip-compression

[14] G-Loaded Journal: Use mod_deflate to Compress Web Content delivered by Apache [online]. Květen 2008.
Dostupné na URL:
http://www.g-loaded.eu/2008/05/10/use-mod_deflate-to-compress-web-content-delivered-by-apache

[15] DevShed PHP Tutorials, Alejandro Gervasio: Output Caching with PHP [online]. Leden 2005.
Dostupné na URL: http://www.devshed.com/c/a/PHP/Output-Caching-with-PHP

[16] The UK Web Design Company, Ross S. McKillop: PHP Caching to Speed up Dynamically Generated Sites [online]. Březen 2004.
Dostupné na URL: http://www.theukwebdesigncompany.com/articles/php-caching.php

[17] The Apache Jakarta Project: Jmeter User's Manual [online].
Dostupné na URL: http://jakarta.apache.org/jmeter/usermanual

[18] Chip's Half Baked Ideas: Using JMeter and Firefox to load test [online]. Leden 2010.
Dostupné na URL:
http://chipcorrera.wordpress.com/2010/01/25/using-jmeter-and-firefox-to-load-test

[19] MySQL: The MySQL Query Cache [online].
Dostupné na URL: http://dev.mysql.com/doc/refman/5.5/en/query-cache.html

[20] MySQL: A Practical Look at the MySQL Query Cache [online].
Dostupné na URL:
http://dev.mysql.com/tech-resources/articles/mysql-query-cache.html

Seznam obrázků

Seznam příloh